Automating the Analysis of Stateful Feature Models

Pablo Trinidad Martín-Arroyo

PhD dissertation

Supervised by

Dr. Antonio Ruiz Cortés

Universidad de Sevilla

December 2012

First published in January 2013 by
Pablo Trinidad Martín-Arroyo

http://www.lsi.us.es/~trinidad
ptrinidad@us.es

Support: PhD dissertation granted by the University of Seville, the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and the Andalusian Government projects ISABEL (TIC-2533) and THEOS (TIC-5906)

A mis abuelos y a Joaquín

Agradecimientos

El desarrollo de una tesis es un camino largo, en el que se encuentran muchos obstáculos, momentos difíciles y momentos de euforia. En este camino he tenido muchos compañeros de viaje que me han empujado y animado a llevar a cabo con éxito esta tesis.

He tenido la suerte de recorrer este camino con Antonio. No puedo entender este trabajo sin su capacidad de sacar siempre algo más y esforzarse al máximo para llegar más allá de donde creemos posible. Este trabajo es fruto de nuestra ambición, que nos ha llevado tal vez más lejos de lo que la costumbre establece pero que esperamos que marque un antes y un después en nuestra vida investigadora. Para él toda mi gratitud y afecto.

Quiero agradecer los ánimos y el cariño que he recibido de aquellas personas con las que comparto mi día a día y que hacen que compensen todas las desavenencias que sufrimos a diario. A Carlos, amigo desde "chiquititos" que me sufre a diario en todas mis versiones; A Sergio, José María, Guti, Cristina y Adela, con quien comparto y disfruto gran parte de mi día a día; A David Ruiz, cuyos consejos me ayudan a mejorar día a día; A David Benavides, mi primer compañero de viaje, que juntos comenzamos a creer en esto de la investigación y con quien comparto muchos éxitos y momentos felices; A Manolo, el mejor compañero en mis aventuras docentes y espero que en un futuro muy próximo también en mis aventuras investigadoras; A Jesús, "mi becario" que me recuerda mucho a cuando era insultantemente joven y tenía energías para comerme el mundo. Espero poder dedicarte todo mi esfuerzo en sacar lo mejor de ti; Y al resto de personas con quien comparto menos momentos pero no por ello menos gratificantes: José Antonio, Kino, Bea, Amador, Octavio, Ana Belén, Pablo, Jose y Sergio Pozo. Por último, y no por ello menos importante, a Miguel Toro porque en dos ocasiones me hizo encontrar la pieza que faltaba para que todo encajara.

Quisiera también hacer una mención especial a los miembros del tribunal, que en los meses previos a la defensa se han prestado a revisar este trabajo y han ofrecido puntos de vista que han ayudado a mejorar este trabajo.

Gran parte del sobreesfuerzo y el desgaste lo sufre mi familia. A mis padres, hermanos, primos y tíos para los que me faltan palabras para agradecerles su cariño y

apoyo. A María, mi principal soporte en este largo camino y que me ha ayudado en los mejores y peores momentos a sacar esto adelante. Y a mis abuelos y a Joaquín, todos los días que habéis faltado siempre habéis estado presentes.

♢♢♢

I'd like to make a special mention to the panel members who, in the months before the defense, have contributed with their valuable revisions, which has helped to improve the quality of this work.

Abstract

Modeling variability is a major task in developing *Software Product Lines (SPLs)*. *Feature Models (FMs)* are the most widely used model for this purpose. A FM represents as a hierarchy of features, the set of decisions that users can take to configure their products. To date, these decisions are limited to select and remove features, preventing decisions on other important elements such as cardinalities and attributes.

Moreover, the automated extraction of information from FMs, a.k.a *Automated Analysis of Feature Models (AAFM)* is a thriving topic that has caught the attention of researchers for the last twenty years. The AAFM offers a wide range of analysis operations for different purposes. The general approach to solve these analysis operations is to give an operational semantics in terms of declarative languages that allow the extraction of information by means of logic solvers. Following this approach over 30 operations analysis have been proposed to date.

A subset of these transactions so-called *explanatory operations* offers the possibility of providing explanations for the relationships that cause certain errors or conflicting user decisions to be repaired in a configuration. However, of all proposed explanatory operations to date, only a subset of them has a formal semantics.

In this scenario there are three problems that this thesis faces: first, FMs are not fully-configurable since they prevent decisions on any kind of element. Second, it is necessary to endow all the explanatory operations with a formal semantics. Third, there is a large number of analysis operations that do not support fully-configurable FMs. It raises a need to propose a new formal framework for their support.

In this work we start from two conjectures: that there is a correlation between explanatory and non-explanatory operations, and it is possible to interpret both types of operations as *Deductive and Abductive Problems (DAPs)*.

Relying on these assumptions, in this thesis we present three main contributions in order to solve the raised problems: (i) we propose *Stateful Feature Models (SFMs)* as fully-configurable models that enable users to make decisions about all of its elements, (ii) the use of SFMs and its interpretation as DAPs allow us to give a formal semantics

for explanatory analysis in a compact manner, performing all the operations proposed to date as special cases of two explanatory operations, (iii) as we propose a new model, we see the opportunity to review the entire catalogue AAFM operations, proposing a simplified catalogue operations and a set of composition mechanisms which give flexibility to define new analysis operations.

With these contributions, we believe that this work sets the basis for the *Automated Analysis of Stateful Feature Models (AASFM)* that solves the limitations identified in this work for the AAFM and simplifies the formalisation process and the implementation and testing of the analysis engines.

Resumen

El modelado de la variabilidad es una de las principales tareas en el desarrollo de *líneas de productos software* (LPS). Los FMs son el modelo más utilizado para ello. Los FMs representan el conjunto de decisiones que pueden tomar los usuarios para configurar su producto como una jerarquía de características. Hasta la fecha, estas decisiones se limitan a elegir y descartar las características que se desean, impidiendo la toma de decisiones sobre otros elementos importantes como son las cardinalidades y los atributos.

Por otro lado, la extracción automática de información de los FMs, también conocida como *análisis automático de FMs* (AAFM) es un tema que ha sido objeto de investigación en los últimos veinte años. El AAFM ofrece un amplio catálogo de operaciones de análisis para distintos propósitos. El enfoque general para resolver estas operaciones de análisis consiste en dar una semántica operacional en términos de lenguajes declarativos que permiten la extracción de información por medio de resolutores lógicos. Siguiendo este enfoque se han propuesto hasta la fecha más de 30 operaciones de análisis.

Un subconjunto de estas operaciones denominadas *explicativas* ofrecen la posibilidad de obtener explicaciones sobre las relaciones que provocan determinados errores o las decisiones de usuario conflictivas que deben repararse en una configuración. Sin embargo, de todas las operaciones explicativas propuestas hasta la fecha, sólo un subconjunto de ellas dispone de una semántica formal.

En este escenario encontramos tres problemas a los que esta tesis se enfrenta: en primer lugar, los FMs no son modelos completamente configurables al impedir la toma de decisiones sobre todos sus elementos. En segundo lugar, es necesario dotar a todas las operaciones explicativas de una semántica formal. En tercer lugar, existe un elevado número de operaciones y la incapacidad de algunas de ellas para trabajar con FMs completamente configurables plantea una necesidad de proponer un nuevo marco de trabajo formal que les de soporte.

En este trabajo partimos de dos conjeturas: que existe una correlación entre determinadas operaciones explicativas y otras no explicativas; y que es posible interpretar

ambos tipos de operaciones como problemas de análisis abductivo y deductivo (DAP).

Apoyándonos en estas conjeturas, en esta tesis presentamos tres principales contribuciones a fin de resolver los problemas planteados: (i) proponemos los SFMs como modelos completamente configurables, que permiten a los usuarios tomar decisiones sobre todos sus elementos, (ii) el uso de los SFMs y su interpretación como DAPs nos permite dar una semántica formal al análisis explicativo de una manera compacta, interpretando todas las operaciones propuestas hasta la fecha como casos particulares de dos operaciones de análisis explicativo, (iii) al proponer un nuevo modelo para el análisis, vemos la oportunidad de revisar todo el catálogo de operaciones del AAFM, proponiendo un catálogo simplificado de operaciones y un conjunto de mecanismos de composición que otorga flexibilidad a la hora de definir nuevas operaciones de análisis.

Con estas contribuciones, entendemos que este trabajo establece las bases del análisis automático de SFMs (AASFM) que resuelve las limitaciones identificadas en este trabajo para el AAFM y que simplifica el proceso de formalización, de implementación y de pruebas de los motores de análisis.

Contents

List of Figures

List of Tables

Part I

Introduction

1

Introduction

The secret of happiness is not in doing what one likes, but in liking what one does.

Sir James Matthew Barrie (1860–1937),
Novelist

This Chapter presents an overview on the results presented through this dissertation. In Section §1.1 we review the research context. In Section §1.2 we motivate this work, explaining its purpose. Section §1.3 details the problem in terms of research questions. Section §1.4 defines several goals for this work. In Section §1.5 we anticipate the approach followed to fulfil the goals. Section §1.6 presents the context in which this dissertation has been carried. Last, the structure of this dissertation is presented in Section §1.7.

1.1 RESEARCH CONTEXT

The construction of products tailored to the specific needs of each customer is one of the challenges that the hardware industry has faced. In the crafts a client is free to customise a product choosing its features, but its cost can skyrocket. In mass production, costs are greatly reduced but the customer loses its ability to make decisions and must submit to the products already manufactured. Product lines are an alternative to achieve the so-called mass customisation, an intermediate point between craft and mass production where customers are allowed to make a set of decisions to customise the final product that best suits their needs within a range of possible features.

Producing customised software product is demanded in many different scenarios. So for example, Android OS must be adapted to work in different mobile devices; or Ubuntu OS that can be adapted to different user needs by offering a wide range of configuration options before and after installation. Developing and maintaining a specific OS for each terminal or user computer is generally an unfeasible approach in terms of cost and time. SPLs [23] incorporate successful practices in hardware product lines to the software world. SPLs propose a set of techniques for the development, maintenance and evolution of customised software products in which the end user plays a key role.

Since the set of products that can be manufactured in a SPL can be huge, with thousands of products [57, 80] it is necessary to have models that make it possible both to represent the complete set of products in a compact manner and to enable its systematic and automated management. *Feature Models (FMs)* [53] are one of the most widely used models for this purpose, proposing a compact representation of all the products in an SPL in terms of their features. A feature is an increment in product functionality [5]. Features are connected by means of relationships among them, forming a tree-like structure. Relationships constrain the way in which features can be combined. Besides features, FMs might use cardinalities to group features in the so-called *Cardinality-Based Feature Models (CBFMs)* and/or attributes to remark non-functional characteristics of products in the so-called *Extended Feature Models (EFMs)*.

The process by which one or more users define the product that best fits their needs by making successive decisions on a particular FM is called *configuration process*. This process is successfully accomplished when a threefold condition fulfils: there are no more user decisions to make, there exists only one product in the SPL satisfying each and every user decision, and there are no contradictions among decisions. The deci-

sions made by users in a configuration process are collected by the so-called *Configuration Model (CM)*, or simply a *configuration*.

The automated extraction of information from FMs and CMs, a.k.a *Automated Analysis of Feature Models (AAFM)*[1] is a thriving topic that has caught the attention of researchers for the last twenty years [7, 14]. It is commonly accepted that this extraction is carried out by the so-called *analysis operations* for which there is already a catalogue with over 30 analysis operations. The analysis operations allow for example to know whether a feature model is valid (i.e. it represents at least one product), how many products a FM represents, whether a CM is valid (i.e. each and every user decision can be made) or to calculate different metrics.

One of the most important applications of the AAFM is the so-called FM debugging [7, 53, 89], a process to produce error-free FMs. Debugging might be performed in three steps: error detection, error explanation and reparation. Detection performs several analysis operations to find different kinds of error. Error explanation identifies the relationships that cause such errors. These explanations are used to assist with the manual or automatic reparation of errors. Besides debugging, explanations have other applications such as explaining the conflicting user decisions in invalid configurations [99] or explaining why a FM has certain properties [85]. In general terms, an explanation can be defined as an analysis operation that takes a FM, optionally a CM and an analysis operation as inputs and returns information (so-called explanations) about the reasons of why or why not the corresponding response of the operation.

The use of declarative paradigms in the AAFM is the main trend, due to the availability of off-the-shelf solvers able to reason about them. Most of the proposals in the AAFM perform analysis operations by means of propositional logic [5, 64], constraint programming [10, 89, 99] or description logic [39] among others. This approach has enabled the development of AAFM engines such as AHEAD [6], FaMa Framework [90], Gears [79], pure::variants [72] and SPLOT [62]. The use of declarative techniques, executable by solvers, endow the analysis operations with an operational semantics [48]. Alternatively, some authors [9, 16, 22] have proposed an axiomatic semantics to prove certain properties of FMs. Durán et al. [36] propose a translational semantics that can be used to check the correctness of the analysis operation implementations [78]. The AAFM in general is a research field with an important focus on formalisation. However, only a few proposals dare to formalise explanations such as [5, 37, 89, 99, 100].

[1] Traditionally the term *configuration model* has been omitted both in the acronym AAFM and in the definition, although it has been always implicitly present.

Despite those formal proposals, there are still many scenarios [85] in which explanations are used where no formal or informal solution has been given, even when there exist tools that support them.

1.2 MOTIVATION

As an initial conjecture of this dissertation, we claim that in general, explanatory analysis is not regarded as a unique operation but it should be actually regarded as a set of related operations that constitutes a parallel hierarchy with respect to the remaining catalogue operations [85]. We refer to this set of parallel operations as *explanatory operations*.

As it is highlighted by Benavides et al. [14], explanatory operations are a challenging operation in the AAFM. In order to provide an efficient tool support, explanations must be as accurate as possible, which in most of the cases means to be minimal. This becomes an even more challenging task when considering CBFMs or EFMs, that we call ECBFMs for short. Furthermore, this last consideration also affects to both non-explanatory analysis of FMs and configuration modelling. Thus, for example, current CMs prevent the representation of decisions such as 'I want my Android OS has at least two Internet connections' or 'the product cost must be less than 200 €'.

A good number of works give a semantics to the AAFM, mapping FMs and CMs into different formal (and informal) languages. According to the quotes received, mapping into *Constraint Satisfaction Problems (CSPs)* is the most widely accepted approach [10]. Probably, the declarative nature of CSPs and the existence of a myriad of CSP solvers are in the root of its success. Unfortunately, the succinctness and declarative style of CSPs is compromised by explanatory operations [89, 99], becoming in an uncomfortable formalism to analyse them [85]. Maybe due to this fact, the formalisation of explanatory operations has not been thoroughly studied, being an important open issue in the AAFM.

As a second conjecture of this dissertation, we claim that it is possible to define an operational semantics in terms of a new declarative language able to deal with explanatory and non-explanatory operations. We have coined this new language as *Deductive and Abductive Problem (DAP)*. As in the case of CSPs, DAPs have a declarative nature and it is easy to devise DAP-solvers with a lot of tools, being CSP one of them.

Finally, analysis engines deal with complex data structures and algorithms (FaMa

Framework contains over 20,000 KLoC). Thus, their implementation is far from being trivial, leading to errors easily, increasing the development time and reducing the reliability of analysis solutions. Gaining confidence in the absence of faults in these tools is especially relevant since the information extracted from FMs is used all along the SPL development process to support both marketing and technical decisions [53].

According to this scenario, our motivation is fourfold. First, assuming that current ECBFMs are not fully-configurable, i.e. current CMs do not enable making decisions about cardinalities (CBFMs) or attributes (EFMs), we are specially interested in offering a solution that extends the structure of current models, and analysing the impact of such extension onto the current analysis operations catalogue.

Second, since explanatory analysis has not been thoroughly studied, we are especially attracted both by finding out how many explanatory operations remain undiscovered and by providing them a semantics as simple, configurable and expressive as possible.

Third, as formal specification frameworks are as difficult to build and taking into account that currently there are more than 40 analysis operations, which 11 of them are explanatory operations, and ECBFMs are not supported, we aim to provide a formal specification framework that overcomes these drawbacks.

Finally, as implementing and maintaining an analysis engine is tedious and error-prone, especially when it has to be gradually updated; we aim to build an analysis engine as a comprehensive, easy-to-maintain reference implementation of the formal specification framework developed in this dissertation. We have a special motivation in achieving a facade of our formal framework in such a way that tool developers do not need to be experts at the underlying formalisms.

1.3 PROBLEM STATEMENT

As outlined in the above discussion, the support of both the automated analysis of FMs and CMs, and the configuration modelling is an emerging research topic, where a number of problems still needs to be considered. In particular, the challenge this dissertation addresses can be stated by the following three research questions:

1. Is it possible to extend CMs to enable fully-configurable ECBFMs?
2. Is it possible to improve the current support for the explanatory operations on

ECBFMs?

3. Is it possible to improve current formal specification frameworks for the AAFM?

1.4 THESIS GOALS

The main goal of this dissertation is to devise a set of solutions that improve the support of both the AAFM and the configuration modelling. To this end we propose for each question a number of subgoals that have guided our research process.

First, regarding **research question 1**, we aim to improve configuration modelling in such a way that enables: *i)* representing user decisions on attributes and cardinalities, *ii)* taking advantage of the information that FMs and CMs have in common such as features, cardinalities and attributes, *iii)* representing CMs together with FMs with a backwards-compatible graphical notation.

Regarding the **research question 2**, we aim to improve the automated analysis of explanatory operations in such a way that enables: *i)* applying explanatory operations to fully-configurable ECBFMs. *ii)* identifying and formally defining the comprehensive set of undiscovered explanatory operations, and *iii)* interpreting explanatory operations as abduction problems (see Section §3.2).

Regarding the **research question 3**, we aim to provide a formal specification framework in such a way that enables: *i)* defining a new, comprehensive analysis operations catalogue encompassing explanatory and non-explanatory operations on fully-configurable ECBFMs, *ii)* identifying a minimum set of primitive operations on the basis of which the remaining analysis operations in the catalogue can be defined, *iii)* interpreting non-explanatory operations as deduction problems (see Section §3.1).

Deduction Problems (DPs) and *Abduction Problems (APs)* are partial abstractions of abductive and deductive reasoning (see Chapter §3). These abstract models provide a simple but expressive enough framework that: *i)* allows to assign a translational semantics to fully-configurable ECBFMs, *ii)* provides utilities to interpret all the analysis operations, and *iii)* can be implemented by a plethora of deductive and abductive solvers that supports a wide range of declarative languages which can be used for the automated analysis in an efficient manner.

1.5 SOLUTION PROPOSAL

The underlying problems of this dissertation goals could be tackled in a number of ways, and in an independent manner, but we conjecture that taking a holistic approximation where they are considered as subproblems of a single, broader problem allows us to provide a more general, simple and scalable solution.

As a first decision of our approximation, we propose to unify FMs and CMs in a new model that we coin as *Stateful Feature Model (SFM)*. On the design of this new model, we aim that any FM (from FODA to ECBFM) can be interpreted as an SFM without any user decision, i.e. on which the configuration process has not started yet. It eases the definition of explanatory and non-explanatory operations which might benefit from new definitions while it still enables a backwards-compatible concrete syntax.

As a second decision and in order to improve the current support of both configuration processes and automated analysis, we propose to review both of them considering that: *i)* they must be applied on SFMs, the new unified model, and *ii)* the analysis operations must be defined solely on DAPs and pursuing a minimum number of primitive operations.

Additionally, our solution will have a *Model-driven Engineering (MDE)* flavour, but it will not be fully-fledged MDE.

1.6 THESIS CONTEXT

This thesis has been developed in the context of the research group Applied Software Engineering (Ingeniería del Software Aplicada-ISA) of the University of Seville, and it proposes a new research topic within the SPL area. These are the research projects that have made the development of this dissertation possible:

- WEB-FACTORIES: In the context of this national project, the explanation of errors is interpreted as a diagnosis problem that can be solved using CSPs. FAMA Framework is developed.
- ISABEL, *Ingeniería de Sistemas Abiertos Basada en LínEas de productos*: In the context of this regional project we propose the automated reparation of invalid configurations following a diagnosis approach. The first catalogue of explanatory

operations is also proposed.

- SETI, *reSearching on intElligent Tools for the Internet of services*: In the context of this national project, FAMA Framework is extended to provide a support of explanatory operations.

- THEOS, *Tecnologías Habilitadoras para EcOsistemas Software*: In the context of this regional project, we propose the use of SFMs and *Automated Analysis of Stateful Feature Models (AASFM)* to save the limitations found in the AAFM regarding the formalisation of explanatory operations.

1.7 Structure of this dissertation

This dissertation is divided in four parts:

Part I. Introduction. This chapter provides an overview of the contributions of this dissertation. Chapter §2 summarises the background concepts in which this dissertation relies on. In Chapter §3 we introduce DAPs as abstractions of a number of well-known techniques in the area of abductive and deductive reasoning.

Part II. Our contribution. Chapter §4 presents the motivation of this work. Chapter §5 defines SFMs, proposing three different representations of them: abstract model, stateful feature diagrams and *Stateful Feature Metamodel (SFMM)*. Chapter §6 provides an overview of the AASFM. Chapter §7 identifies the realisation of query operations as the resolution of DPs. Chapter §8 identifies the realisation of explanatory operations as the resolution of APs. Finally, Chapter §9 proposes a catalogue of operations for the AASFM based on previous results.

Part III. Verification of Results. Chapter §10 describes a prototype tool that has been built using model-driven engineering to test the results presented in this dissertation.

Part IV. Final Considerations. Chapter §11 briefly revises the contributions in this dissertation and explores the opportunities that arise from this work.

Three appendixes are also provided as additional resources that have been generated throughout the development of this dissertation:

Annex §A. The design documents that have been generated for the design of the stateful feature metamodel are presented in this annex.

Annex §B. The document presented in this annex has been used to interpret a SFM in terms of constraint programming to build a prototype of the results presented in this dissertation. It also discusses the use of other logic languages that can be used to implement the AASFM.

Annex §C. It provides a brief summary of *First Order Logic (FOL)*.

2

Background Information

Humour is part of life, and therefore should not be shut out even from serious literature

Lin Yutang (1895–1976),
Chinese writter

In this Chapter we present the background concepts that are provided for a better understanding of the problems and solutions described in this dissertation. First, we present FMs in Section §2.2. Section §2.3 describes the configuration process and a general configuration model. Section §2.4 presents the AAFM as a set of techniques for the automated extraction of useful information from FMs. To illustrate its scope, some of the most used analysis operations and the general approach to implement AAFM operations are presented. In Section §2.5 we focus on the state of the art of explanatory analysis, a remarkable subset of AAFM operations. Section §2.6 provides an overview on constraint satisfaction problems which are a widely used approach in the AAFM. Finally, Section §2.7 summarises the background concepts and advances their relationship with this dissertation.

2.1 SOFTWARE PRODUCT LINES

The mass production aims to reduce the costs in the production of a large amount of hardware products. Unitary costs are reduced while the benefit margins increase, enabling the investment in technological improvements that improve the quality of the products. A mass production produces identical products which may not fit into user needs, and therefore limiting the target market.

Adapting a product to user needs is usually considered a craftwork whose costs are quite higher than a massive product. The mass customisation aims to be competitive in the production of customised products. A user is given a margin of freedom in their decisions so that products are customised while the benefits of a mass production are still kept. It allows to produce customised products at very competitive costs.

The digital distribution of software has almost buried the physical distribution and therefore the mass production of software. However the mass customisation in software at a low cost and time-to-market is still an open-issue. The *Software Product Line (SPL)* approach [23] offers a set of techniques, methods and methodologies to produce customised products in mass. It is based in the similarity and segmentation among products. So products are produced in a specific domain where the commonalities and variabilities among demanded products is well-known.

SPLs systematise the reuse of software, building a common base of assets that are later combined to generate products that are adapted to user needs. This approach is opposite to *ad-hoc* reuse which searches for reusable software pieces from other already-built products to reduce the time to market.

2.2 FEATURE MODELS

One of the main challenges in SPLs is about the definition of all the possible products that a user can choose. Since the number of products can be very large, it is necessary to represent all of them compactly. One of the most used techniques to represent all the products within a SPL are the so-called *Feature Models (FMs)*. Products are described in terms of features, which are distinctive characteristics a user can observe [53]. A FM can de defined as follows:

Definition 2.1 - Feature Model.
A FM is a tuple (F, P) such that F is the set of features in the SPL and P the set of products defined as a subset of the powerset of features, i.e. $P \subseteq \mathcal{P}(F)$, such that a product is defined by a set of features in F.

Let us take a SPL of *Smart Home Systems (SHSs)* as an example. Its set of features F and a possible product P_1 (see Figure §2.1) can be defined as follows:

$$\begin{aligned} F = \; & \{SmartHome, Lighting, ControlSystem, CellPhone, ControlPanel, \\ & AntitheftAlarm, Internet, Ethernet, 3G, WiFi-b/g, WiFi-n, MoviePlayers, \\ & HDTV42, HDTV32, PCPlayer, Contents, VideoOnDemand, Providers, \\ & Cache, DMS\} \\ P_1 = \; & \{SmartHome, Lighting, CellPhone, Alarm, HDTV42, \\ & HDTV32, DMS, VideoOnDemand\} \end{aligned}$$

In order to define the set of products, a FM comprises a set *relationships* that limits the allowed feature combinations, so that a product must satisfy all the relationships. Relationships in a FM are mainly hierarchical. Any FM has a *root* feature that represents the whole functionality of any product. The root feature is refined in child features, which decompose the behaviour or functionality of the root feature into sub-features, which describe the scope of the root feature in more detail. This refinement process is repeated for the child features to conform a tree-like structure. Although the hierarchical structure helps to represent the feature refinement, it can hinder the representation of restrictions that affect features in different branches of the tree. In these circumstances, *cross-tree constraints* can be used.

Feature diagrams [75, 76] are probably the most used graphical representation of FMs. Figure §2.2 presents a feature diagram for a SPL of SHSs.

The so-called basic FMs have evolved in time adding new elements to the set of features and relationships. So Czarnecki et al. [29] and Riebisch et al. [75] propose CBFMs as an evolution of basic FMs that introduce cardinalities. They allow to group a set of features and assign them a cardinality that denotes the number of features (cardinal) that can be selected at the same time. CBFMs increase the succinctness of the model, and they are as expressive as basic FMs, as Schobbens et al. [76] proved.

CBFMs use the following kinds of hierarchical relationships in FMs:

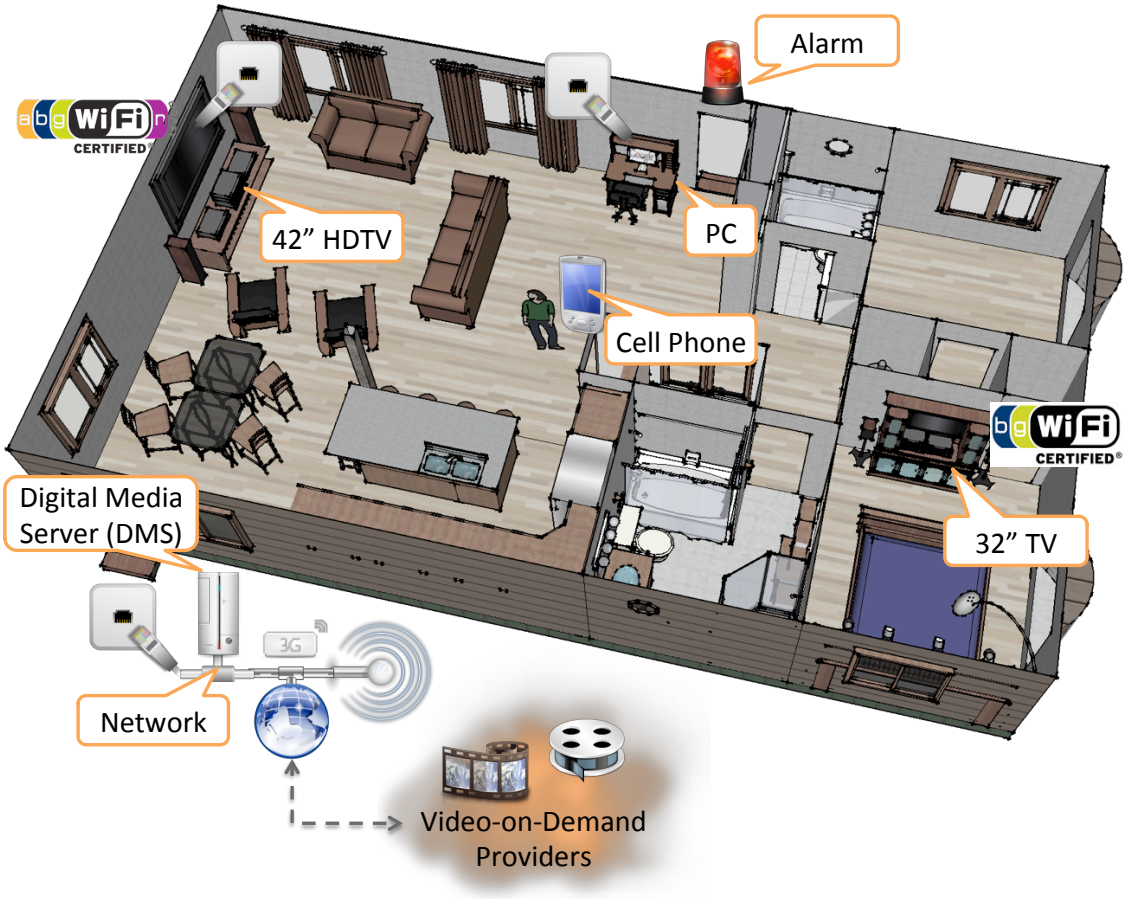


Figure 2.1: An example of a product built from a smart home SPL

- **Mandatory**: a mandatory relationship affects a parent and child feature. It forces the child feature to appear in a product whenever its parent feature does. For example, any SHS must have `lighting` and `controlSystem` features. If a `videoOnDemand` feature is selected, then `providers` must also be selected.

- **Optional**: a child feature connected to a parent feature by means of an optional relationship may be optionally selected whenever its parent feature is. For example, the `antitheftAlarm` and `Internet` connection are optional features in a SHS.

- **Set relationship**: set relationships affect a parent feature and a set of two or more child features. It contains a set of natural numbers or *cardinality* that constraints the number of child features to be selected in a product whenever its parent feature is selected. For example, if the *Internet* feature is selected, then 1 to 4 features must be selected from `Ethernet`, 3G, `WiFi-b/g` and `WiFi-n` features.

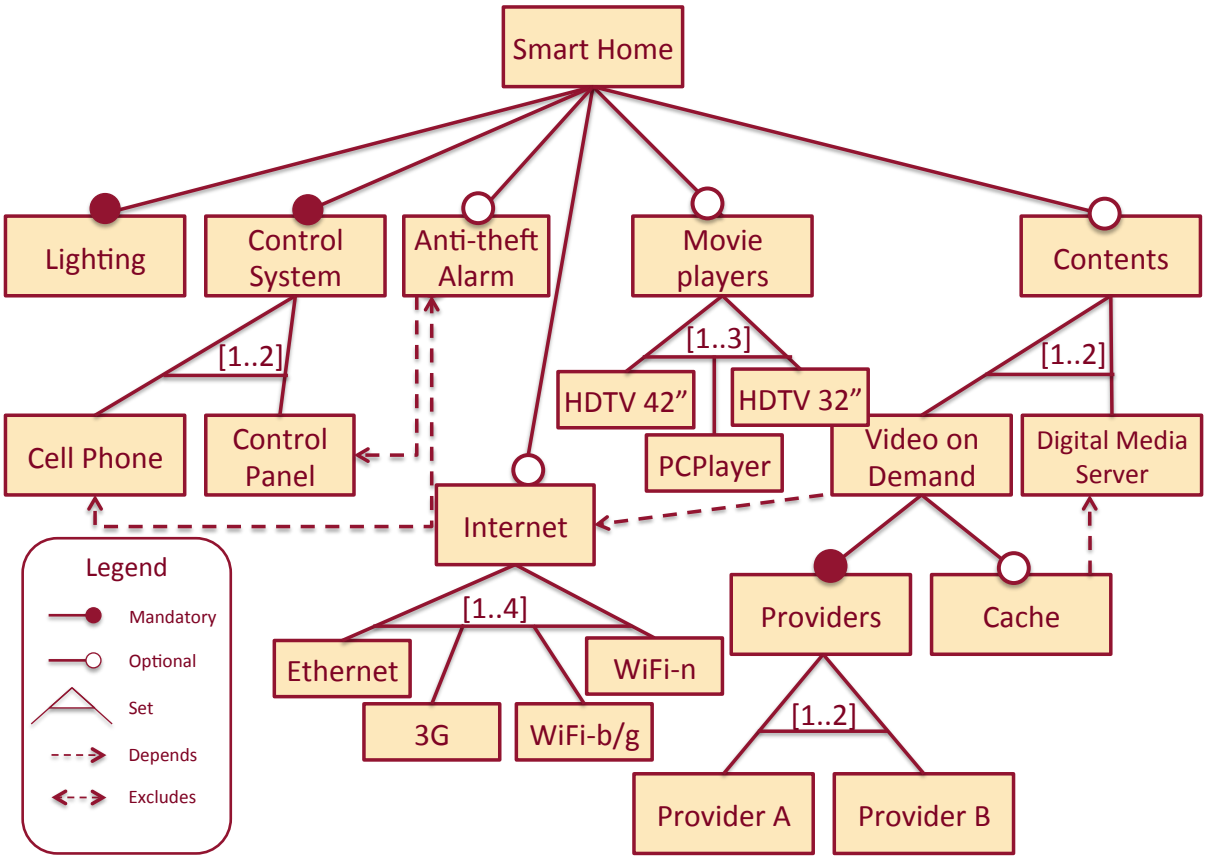


Figure 2.2: A feature diagram example representing a smart home software system

Alternative relationships can be interpreted as a particular case of set-relationship with a [1..1] cardinality, where only one child feature may be selected in a product at the same time if the parent feature is selected. In turn, *or-relationships* are those set relationships whose cardinality is [1..N] such that N is the number of child features. For example, the `InternetConnection` feature is the parent in an or-relationship.

Besides hierarchical relationships, *cross-tree constraints* break the tree-like structure to represent non-hierarchical relationships. The most used cross-tree constraints are:

- **Dependency**: a feature depends on another feature if the second one must be part of a product whenever first one is selected. For example, the `cache` feature requires for a `digitalMediaServer` feature to store video and the `antitheftAlarm` requires a `controlPanel` for de/activation.

- **Exclusion**: two features exclude themselves if both of them cannot be part of a

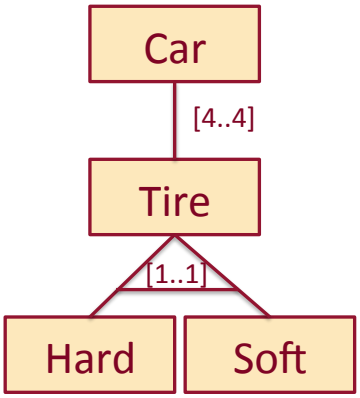


Figure 2.3: Feature cardinalities may lead to ambiguous situations

product at the same time. For example the `antitheftAlarm` feature is incompatible with a `cellPhone` feature for security reasons.

Riebisch et al. [75] also propose a feature cardinality relationship, where a parent and a child feature are linked by a cardinality which indicates the number of valid instances of the child features that can be selected in a product. The cardinality [0..1] is equivalent to an optional relationship and [1..1] to a mandatory relationship. Features in a FM are unique, which means that no other feature can refer to the the same functionality. However, with feature cardinalities it is possible to create more than one feature instance, although the concept of instance still has an ambiguous interpretation. The FM in Figure §2.3 shows an example where the ambiguity arises. A car must have 4 tires, each of which can be hard or soft. Must all the tires be either soft or hard? Or is it possible to combine them anyhow? How do we distinguish between rear and front tires in case we want to combine hard and soft tires? Due to the ambiguity that arises from the use of feature cardinalities, they are usually avoided in CBFMs.

Besides features, FMs can collect additional information by using the so-called *attributes*. An attribute represents relevant information such as feature development cost, versions, RAM consumption, performance or technological requirements. FMs that use attributes are known as EFMs [10]. Figure §2.4 shows an example of a FM with attributes. It adds information regarding Internet bandwidth to an excerpt of the FM in Figure §2.2. Each kind of connection provides a different bandwidth. Since more than one connection can be chosen, the maximum available Internet bandwidth in the SHS is the maximum bandwidth provided by each chosen connection.

An EFM may contain constraints that affect attributes which reduce even more the set of products an EFM describes. So for example, if a constraint sets the `Internet` max

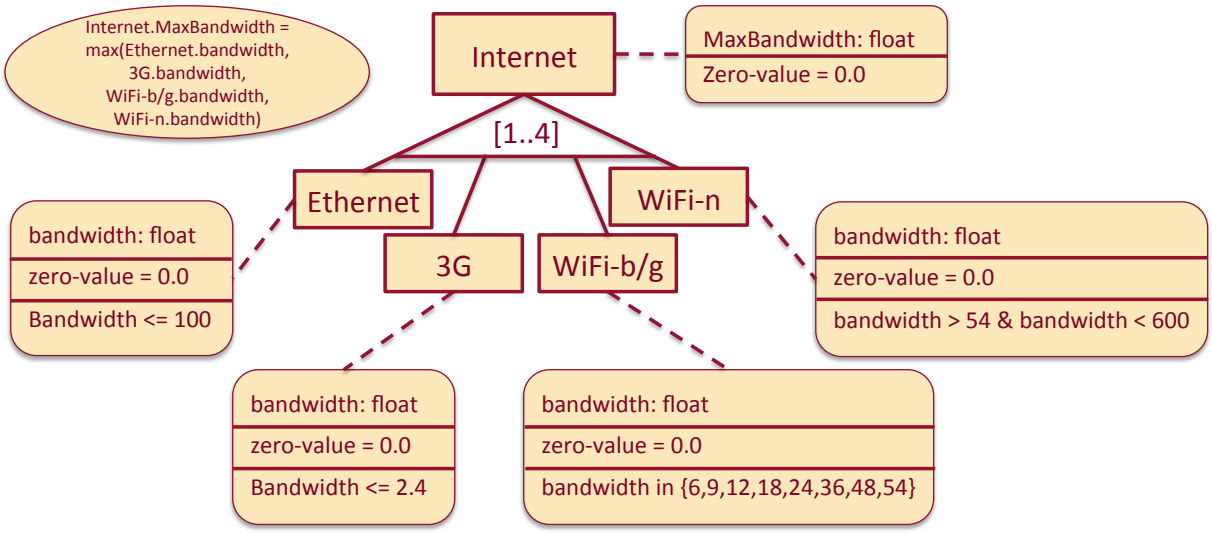


Figure 2.4: An excerpt of a feature diagram representing an extended FM

bandwidth to at least 80 Mbps then it forces any SHS to have at least an `Ethernet` or `Wifi-n` connection since they are the only features providing such bandwidth.

In this dissertation we coin the term *Extended Cardinality-Based FM* (ECBFM) as an FM that incorporates all the capabilities of CBFMs and EFMs. Henceforth, we use the term FM to refer to an ECBFM for the sake of simplicity unless it is necessary to distinguish among them.

2.3 PROCESSES AND MODELS OF CONFIGURATION

The process by which one or more users define the product that best fits their needs by making successive decisions on a particular FM is called a *configuration process*. The decisions made by users in a configuration process are collected in a *Configuration Model (CM)*, or simply a *configuration*. Depending on how users organise the decisions, three kinds of configuration processes are considered:

- **Individual configuration**: only one user participates in the configuration process. A new decision cannot contradict any decision that has been previously made by the same user. Thus, a feature that is already selected cannot be removed later.
- **Staged configuration** [29]: more than one user performs the configuration process in stages. At each stage one user makes decisions, updating the CM. It is

possible that a user wants to change a decision on a feature that has already been selected or removed by another user in a previous stage. In this sense, the contradiction must be annotated in the CM so that the user that selected or removed the features previously is informed so that the conflict can be solved among users.

- **Parallel configuration** [99]: more than one user makes decisions at the same time and in any order, enabling users to contradict each other. So for example, a user might remove a feature that is being selected by another user at the same time.
- **Collaborative configuration** [63]: this process lies halfway between staged and parallel configuration processes. A number of users are allowed to make decisions in parallel but only on a specific part of the FM. These parts are previously calculated so that conflicts among decisions are avoided.

Users usually express their decisions in terms of feature selections or removals, i.e. which features must be part of a product or left aside. In order to collect these decisions CMs are structured as follows:

Definition 2.2 - Configuration model.
Given a FM as a tuple (F,P), a configuration for this FM, denoted as γ_{FM} is a three-tuple of the form (S,R,U) in which S, R and U denote three disjoint finite sets of selected, removed and undecided features respectively in such a way that all the features of F must belong to one and only one of these three sets, i.e.

$$\gamma_{FM} = (S,R,U) \Leftrightarrow F = S \cup R \cup U \text{ and } S \cap R \cap U = \emptyset$$

When a configuration process starts, all the features are in the undecided set to indicate that no decision has been made about them. Whenever a user selects a feature, it is moved from the undecided set to the selected set. If a feature is discarded then it is moved from the undecided set to the removed set. Depending on the distribution of features among the three sets, we can define a configuration state as follows:

Definition 2.3 - Configuration states.
A configuration γ_{FM} is *partial* if there are still decisions to be made, otherwise it is said to be a *full* configuration. Both states are denoted as $partial(\gamma_{FM})$ and $full(\gamma_{FM})$.

$$\begin{aligned} partial(\gamma_{FM}) &\Leftrightarrow U \neq \emptyset \\ full(\gamma_{FM}) &\Leftrightarrow U = \emptyset \end{aligned}$$

Whether partial or full, a configuration is said to be *valid* if there is at least one product in the corresponding FM that contains all the features in the selected set and no feature in the removed set; otherwise it is said to be an *invalid* configuration. Both states are denoted as $valid(\gamma_{FM})$ and $invalid(\gamma_{FM})$.

$$valid(\gamma_{FM}) \Leftrightarrow \exists p \in P \cdot S \subseteq p \wedge R \cap p = \emptyset$$
$$invalid(\gamma_{FM}) \Leftrightarrow \neg\exists p \in P \cdot S \subseteq p \wedge R \cap p = \emptyset$$

From the above definitions, we have an alternative way to consider a product of a FM, namely:

Definition 2.4 - Product.
Any valid full configuration γFM defines a product that only contains the selected features. We denote this product as $product(\gamma_{FM})$:

$$product(\gamma_{FM}) = S \Leftrightarrow full(\gamma_{FM}) \wedge valid(\gamma FM)$$

For example, the following configuration for a SHS is partial since some features have already been selected, some others have been removed and others are still to be decided what to do with them:

$$S = \{SmartHome, Lighting, ControlSystem, CellPhone, Internet, Ethernet\}$$
$$R = \{ControlPanel, MoviePlayers\}$$
$$U = \{Antitheft, Contents, VideoOnDemand, DMS, ...\}$$

Full configurations are the result of a completed configuration process. For the SHS example, the following configuration is full:

$$S = \{SmartHome, Ligthing, ControlSystem, CellPhone, Internet, Ethernet\}$$
$$R = \{ControlPanel, MoviePlayers, Antitheft, Contents, Videoondemand, ...\}$$
$$U = \emptyset$$

The following example describes a valid partial configuration for the SHS SPL since there exists at least one product with the selected features and without the removed features:

$$S = \{SmartHome, Lighting, ControlSystem, Cellphone, Internet, Ethernet\}$$
$$R = \{ControlPanel, Antitheftalarm, Contents, ...\}$$
$$U = \emptyset$$

Since the above example represents a valid and full configuration, it can be said that it represents a product. Products are usually represented in terms of their selected features. For instance, $P = \{SmartHome, Ligthing, ControlSystem, CellPhone, Ethernet, Internet\}$ is the description of a product for the above example of configuration.

The following example describes an invalid configuration since *anti-theft alarm* depends on a *control panel* feature, which is removed:

$$S = \{SmartHome, Lighting, ControlSystem, Cellphone, AntitheftAlarm\}$$
$$R = \{ControlPanel\}$$
$$U = \{Contents, Internet, ...\}$$

Both, valid and invalid configurations can be defined in terms of relationships satisfiability. A relationship is said to be satisfied by a configuration if the selected and removed features correspond to the relationship expected behaviour. In those terms, a valid configuration is the one that satisfies all the constraints, while an invalid configuration violates at least one configuration in the FM.

It is frequent in staged and parallel configuration processes to allow users to contradict previous decisions. In this case, features can be selected and removed at the same time, i.e. the set of features in conflict (C) is $C = S \cap R$. These conflicts must be solved in order to accomplish the configuration process, existing several proposals to repair them [67, 68, 99].

2.4 AUTOMATED ANALYSIS OF FEATURE MODELS

The automated extraction of information from FMs and CMs, a.k.a *Automated Analysis of Feature Models (AAFM)* is a thriving topic that has caught the attention of researchers for the last twenty years [7, 14]. It is commonly accepted that this extraction is carried out by the so-called *analysis operations* for which there is already a catalogue with over 30 analysis operations. The analysis operations allow for example to know whether a feature model is valid (i.e. it represents at least one product), how many products a FM represents, whether a CM is valid (i.e. each and every user decision can be made) or to calculate different metrics. These and more examples of analysis operations are shown in Section §2.4.1. The general schema to carry out these operations is discussed in Section §2.4.2. We give an overview of the available tools for the AAFM in Section §2.4.3.

2.4.1 Analysis operations

Benavides et al. [14] propose the most complete catalogue of operations up to now. From this catalogue we select the most representative operations to illustrate the purpose of the AAFM.

Void feature model

This operation takes a FM as an input and returns a value reporting whether the FM is void or not. A FM is void or invalid if it represents no product at all; otherwise a FM is valid if it describes at least one product. Void FMs are caused by contradicting relationships that impede the definition of any valid full configuration. Figure §2.5 shows an example of this operation validating the SHS FM.

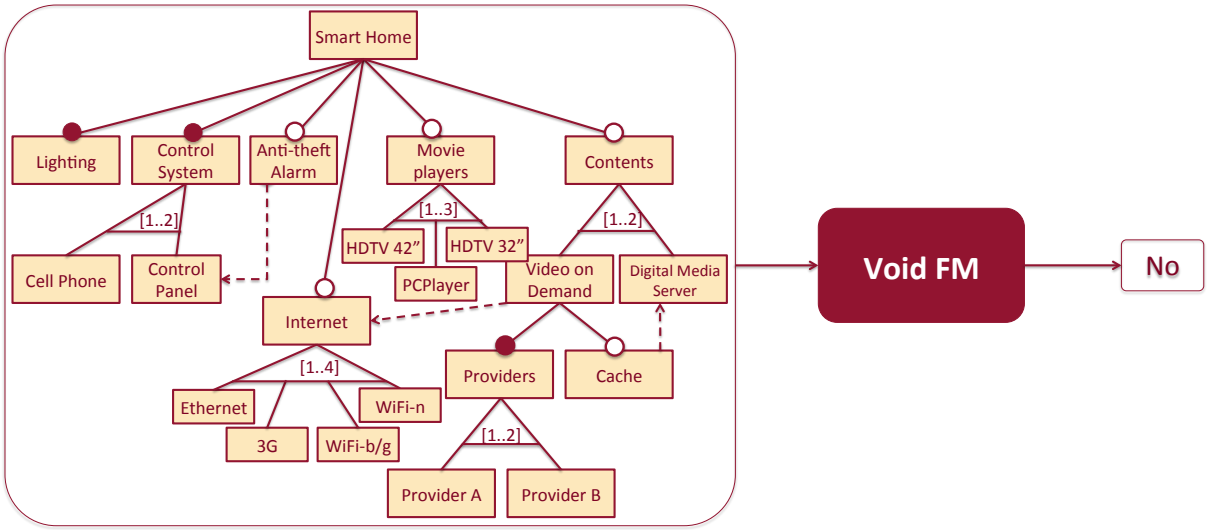


Figure 2.5: An example of a validation operation of the SHS FM

Valid configuration

This operation takes a FM and a CM as inputs and detects if that configuration is valid or not. Figures §2.6 and §2.7 show examples of this operation with a valid and invalid configuration respectively.

Configuration explanation

If a configuration is invalid, it is important to determine which are the decisions that cause it. This operation takes a FM and a CM as inputs and returns one or more explanations in terms of the user decisions that must be undone or corrected in order to restore the configuration validness. Figure §2.7 shows an example this operation

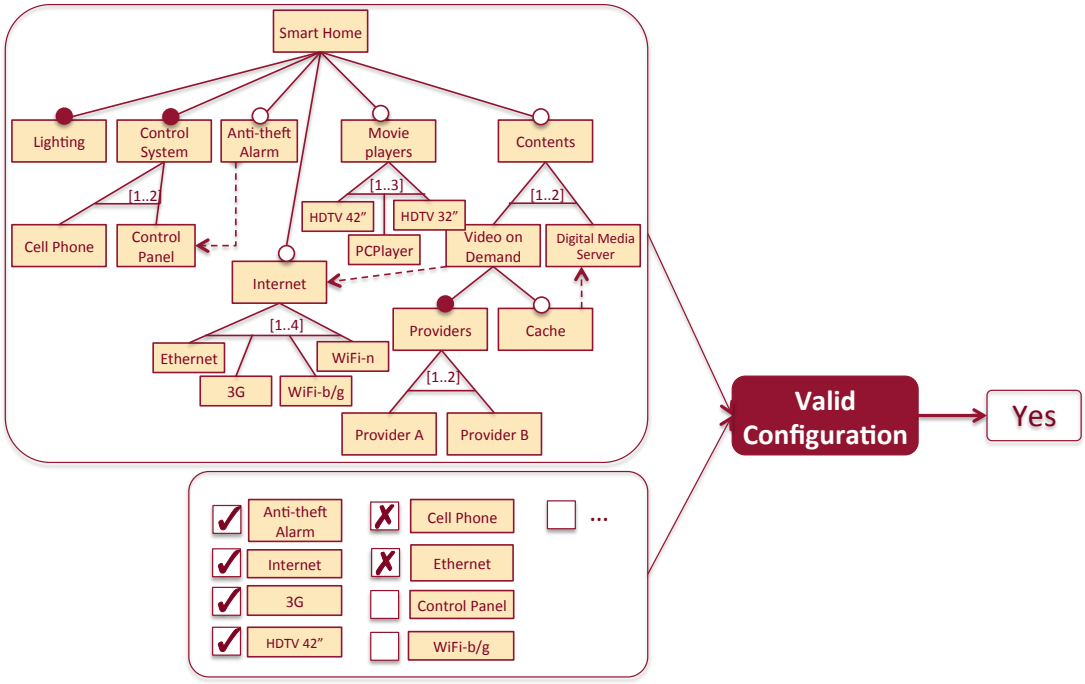


Figure 2.6: An example of operation for configuration validation

for an invalid configuration. The explanations obtained in the example assist the user in changing the configuration, either selecting the `control panel` feature that was removed, or selecting the `antitheftAlarm` and removing the `cellPhone` features.

Error detection

The error detection, a.k.a. anomalies detection is one the three steps in FM debugging. Debugging is a process to produce error-free FMs in three steps: detection, explanation and reparation. The term error refers to unexpected behaviours in a FM that can be caused by mismodelling. Specifically, the errors defined up to date are dead features, false-optional features and wrong cardinals. Figure §2.8 shows a FM example where all these errors appear. Next, we present some of the most used operations to detect errors in FMs:

- **Dead features detection** This operation takes a FM and a feature as inputs and returns a value indicating whether the feature is dead or not. A feature is dead if there is no product where it is selected, i.e. it must be removed in any valid configuration.

- **False-optional features detection** This operation takes a FM and a optional feature as inputs and returns a value indicating whether the feature is false-optional

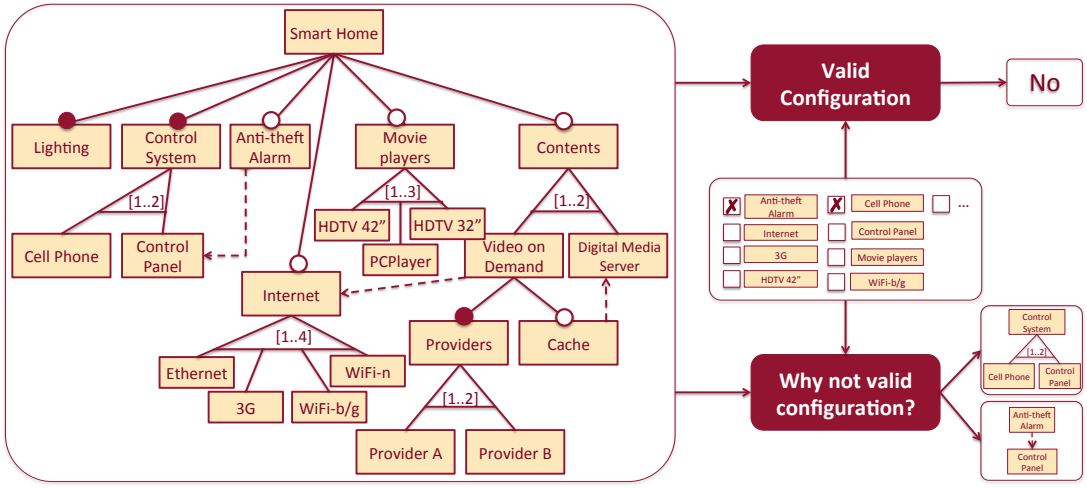


Figure 2.7: An example of the configuration validation and explanation operations

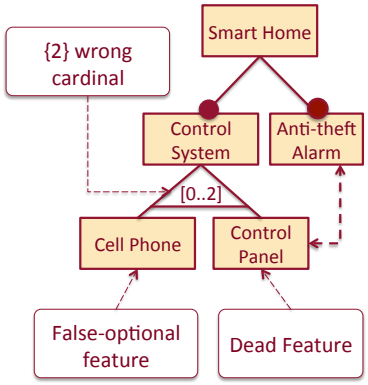


Figure 2.8: Example of anomalies in a FM

or not. A feature is optional if it is the child in a non-mandatory relationship (optional or set). An optional feature is false-optional if whenever its parent feature is selected so the child feature must be. False-optional features must not be confused with core features, which are features that appear in every product. The main difference is that a core feature cannot be removed while a false-optional feature can be removed if the parent feature is also removed.

- **Wrong cardinals detection** This operation takes a FM and a cardinal in a set relationship and returns a value indicating whether the cardinal is wrong or not. A cardinal is wrong if that number of child features in its set-relationship cannot be given for any product defined by the FM.

Relationship explanations

A relationship explanation operation explains why a result is obtained by another analysis operation, such as an anomaly that is detected or a void FM. Explanations are expressed in terms of the relationships in the FM that cause the result. Figure §2.9 shows an example that explains why a feature `controlPanel` is dead due to relationships `R2` and `R4`.

Relationship explanation is not an operation but a set of them. We propose a catalogue of 11 relationship explanation operations in [85], where explanatory operations are divided into 'why not?' operations to explain errors and 'why?' to explain correct behaviours.

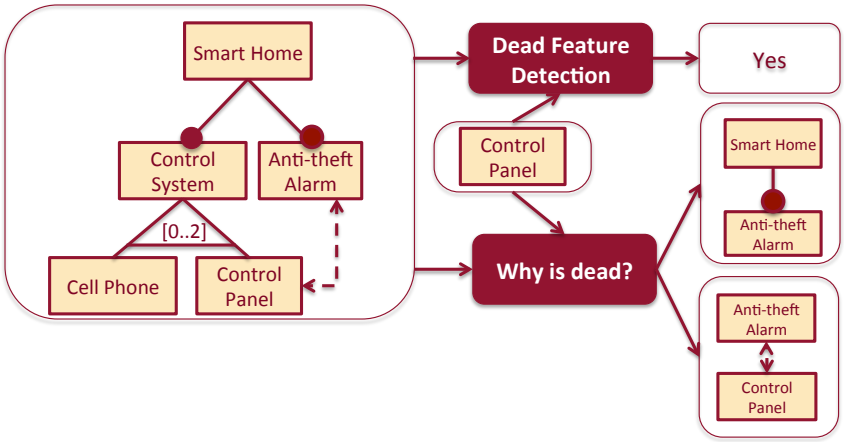


Figure 2.9: Example of dead feature detection and explanation

Many other analysis operations have been proposed in the literature for purposes such as product listing and counting, product optimisation and obtaining metrics. A complete catalogue of analysis operations can be found in [14]. It is not the purpose of this Chapter to present the complete catalogue but to provide an overview on the most relevant operations for this dissertation.

2.4.2 AAFM general schema

The general approach to perform the AAFM is transforming a FM into a declarative knowledge representation that can be used by existing tools or *solvers* for the extraction of information by automatic means (Figure §2.10). The most used representations are:

- Propositional logic: a FM is mapped into a propositional formula that can be evaluated if it is true or false using SAT solvers [15]. Binary Decision Diagrams

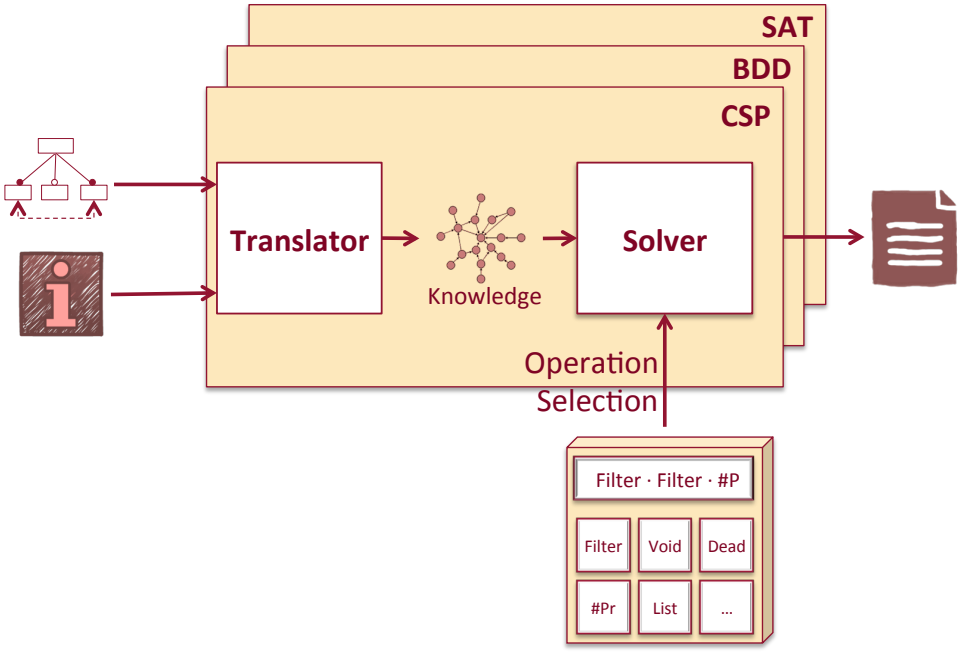


Figure 2.10: General schema of the AAFM

(BDD) [97] are tree-like structures built from a propositional formula that represent the decisions that can be made by users in a configuration process. SAT and BDD solvers have been used for the AAFM by Batory [5], Benavides et al. [12], Czarnecki and Kim [27], Mendonça et al. [64] and Zhang et al. [103].

- Constraint programming: a FM is mapped into a CSP so a constraint solver can be used to perform different analysis operations. We pioneered this approach in 2005 [10] due to its ability to work with non-boolean attributes. Our efforts in the last years have focused on contributing to the use of constraint programming for the AAFM [89, 90, 99].

- Description logic: a FM is mapped into a description logic, which is a representation that describes the knowledge in terms of concepts, roles and individuals [2]. Wang et al. [96] map a FM into a OWL-DL[8], a realisation of description logic in terms of ontologies that are analysed by means of RACER tool [45].

- Clausal logic: a FM is maped into a clausal logic. the most used tool to reason on clausal logics [40] is Prolog [50]. Kang et al. [53] proposed using Prolog for some basic AAFM operations 22 years ago, which is considered the first contribution for the AAFM.

- Others: a minority of works have proposed other mappings such as Zhang et al.

[101, 102] that map FMs into SMV model checker. Some authors have also proposed ad-hoc algorithms to solve specific analysis operations [3, 47, 87, 94, 95].

Each representation enables a subset of analysis operations, each of which having a different performance [12]. So for example a representation can be more suitable for product counting while its performance decreases for product listing.

2.4.3 Tools for the automated analysis of feature models

There exist many SPL development tools that use the AAFM such as CaptainFeature [55], Feature Model Plugin [28], XFeature [18], featureIDE [54] and pure::variants [72]. However none of them was conceived as a general-purpose AAFM tool but used it as a means to solve other SPL problems such as product configuration, feature modelling or product derivation.

As the AAFM and the tools using it grows, the need of a specialised AAFM tool also increases. Trinidad et al. [90] build FAMA Framework (FW)[1], the first general-purpose tool for the AAFM whose main goal is to provide for an implementation of the state of the art techniques and algorithms for the AAFM. FAMA FW is an open-source SPL of AAFM tools where research results can be added as new features. As an SPL, a user can define a custom product selecting the desirable features for a AAFM tool.

SPLOT [62] is a web application for the AAFM presented in 2009. It is a FM repository that allows their edition, analysis and product configuration. It relies on SAT solvers to analyse the anomalies such as void FMs and dead features, and BDD solvers for metrics calculation.

2.5 Explanatory analysis

Debugging FMs and CMs is considered to be one of the main challenges in the AAFM [7, 14]. The goal of debugging is detecting, explaining and repairing any error in FMs and configurations. In error detection, several analysis operations are used to detect errors such as dead features, false-optional features or wrong cardinalities; in error explanation, we obtain the possible reasons why errors have been detected; in error reparation, errors are removed by manual or automated means, assisted by the

[1] http://www.isa.us.es/fama

information provided by the explanations. Many authors [37, 64, 86, 93, 100, 101] have dealt with error detection. However, the explanation and further reparation of errors still presents many open issues [85].

But explanations are not only regarded to errors, but to explain other situations such as invalid configurations, or explaining why a feature is core or variant or why a product is optimal for a given criterion [85]. We coin the term *explanatory analysis* to refer to all the operations that provide explanations for any given situation.

Depending on the model from which we want to obtain explanations, there are two main kinds of operations in the explanatory analysis: configuration and relationships explanation. The solutions proposed for both operations are different. In Section §2.5.1 we overview the proposals that deal with configuration explanation. In Section §2.5.2 we overview those that deal with relationships explanation.

2.5.1 Current support for configuration explanations

Configuration explanations provide the reasons why a configuration is invalid in terms of the user decisions that are provoking the invalidness. These explanations can be used to correct the configuration manually or automatically which should lead to a valid configuration.

Czarnecki et al. [29, 30] are the first authors in proposing the staged configuration process. In every stage in this process, a new FM, named as specialised FM, is generated to reflect all the user decisions that have been made. Moreover, they propose maintaining a CM where user decisions are explicitly collected. Both models, CM and specialised FMs are two sides of the same coin. While users cannot make decisions about cardinalities, cardinalities are only affected by user decisions indirectly when decisions are made on features. The strategy for dealing with errors in the configurations is to avoid situations in which user can make conflicting decisions. They propose a tool that is able to give explanations on conflicting configurations, but no details are given on its implementation,

White et al. [98][99] interpret the configuration explanations as a diagnosis problem, providing an implementation using constraint programming. Besides obtaining explanations, the proposed solution is able to suggest reparations to restore the validness of a configuration. It provides a prototype tool that supports parallel configuration processes.

Mendonça et al. [63] introduce the concept of collaborative configurations. Conflicts might arise when all the user decisions are collected, so it proposes algorithms to divide the FM in safe subtrees each of which is assigned a different user and where decisions can be made without conflicts. They ensure the correct merging of the different parts in which the decision process is divided.

Elfaki et al. [37] interpret FMs as a set of Prolog rules that can be used to support the configuration process, providing for explanations whenever an invalid configuration is found. The proposed algorithms suggest the user decisions that must be removed to repair an invalid configuration. This approach only supports individual configuration processes.

Bagheri et al. [4] propose an algorithm to find the minimal sets of conflicting user decisions in configurations. However, they only provide one solution instead of all the possible solutions. Although it is not explicitly remarked, this approach could be used in parallel configuration processes.

Despite the work of Nöhrer and Egyed [67] does not propose specific solutions to provide explanations, it is worth making a mention to their approach. They propose to allow users to continue making decisions despite an invalid configuration is detected. They suggest to keep on collecting as many user decisions as possible in order to increase the quality of the explanations.

Table §2.1 analyses several aspects of all these proposals. First, we remark if these proposals provide a formal semantics and/or an implementation of the solution in a tool or prototype. Second, the repairing strategies to deal with conflicting configurations is remarked, distinguishing among three strategies:

- Avoiding any situation that might lead to a conflict among user decisions. The main complexity on this approach resides on the difficulty of ensuring that there is no conflicting situation that might arise due to unexpected scenarios in a configuration process.
- Explaining the source of a conflict. In this case, users are allowed to make decisions despite of leading to contradictory configurations.
- Repairing conflicts. This approach suggests ways of automatically repairing a configuration instead of reporting the users about the conflictive decisions.

Third, the kind of input FM and the elements on which users can make decisions

are remarked. All the proposals work with CBFMs and they only permit users to make decisions on features, giving no support for decisions on cardinalities or attributes in user decisions. Last, the kind of configuration process they assume is remarked, either individual, staged, parallel or collaborative.

Configuration explanation†						
Work	**Sem.**	**Impl.**	**Contrad.**	**Model**	**Dec.**	**Process**
Bagheri et al. [4]	+	-	Explain	CBFM	F	Staged
Czarnecki et al. [29, 30]	-	+	Avoid	CBFM	F	Staged
Mendonça et al. [63]	+	+	Avoid	CBFM	F	Collaborative
Nöhrer and Egyed [67]	-	-	Explain	N/D	N/D	Staged
Elfaki et al. [37]	+	+	Repair	CBFM	F	Individual
White et al. [99]	+	+	Repair	CBFM	F	Parallel

† + Supported - Unsupported N/D Undefined

Table 2.1: Current support for configuration explanations

2.5.2 Current support for relationships explanations

Relationships explanations provide the reasons why given a FM, an input configuration is valid or invalid in terms of relationships in the FM [85]. If the input configuration is invalid, it is the FM which is supposed to contain errors so explanations are obtained in terms of the relationships in the FM that make the configuration invalid. They are known as a *'why not?'* operations. If the input configuration is valid, explanations are obtained in terms of the minimal set of relationships that make the configuration valid. They are known as *'why?'* operations.

The explanations provided by 'why not' operations might assist a manual reparation of a FM. Unlike configuration explanations, the automatic reparation of FMs is still an open issue, maybe due to the complexity of FM relationships compared with user decisions in CMs where features can only be selected or removed.

Kang et al. [53] firstly proposed the need to debug FMs using logical representations, specifically Prolog. However they proposed no specific algorithm or technique to deal with explanations.

Batory [5] proposes to use truth maintenance systems to find relationships in a FM that make a configuration to be invalid.

Wang et al. [96] represents a FM into Alloy Analyser, using its ability to obtain explanations in unsatisfiability cases. It can be used to explain configurations and the FM itself in terms of the failing relationships. [81] interprets a FM as a description logic and uses OWL reasoners capabilities to obtain explanations. In both cases, neither a semantics is given nor implementation details are provided.

Trinidad et al. [89] propose the first work where explanations are dealt from a formal point of view. They endow explanations with an axiomatic semantics, interpreting void FMs, dead features and false-optional features in terms of theory of diagnosis. They also propose an operational semantics in terms of CSPs. These operations are later supported by FAMA FW [90] tool suite. In the last years, FAMA FW has incorporated new explanatory operations into its catalogue, providing explanations for wrong cardinalities, optimisation and invalid configurations.

Trinidad and Ruiz-Cortés [85] propose the most extensive catalogue of explanatory operations up to date. 7 'why not?' and 4 'why?' operations are proposed, but no clear semantics is given.

van den and Galvão [93] represent FMs as generalised feature trees, proposing ad-hoc algorithms to obtain explanations for void FMs and dead features.

Zaid et al. [100] propose yet another work that interprets a FM in terms of a description logic to provide explanations on void FMs.

Table §2.2 summarises all these proposals from the point of view of the operations they support from the catalogue in [85]. For each operation, we analyse three factors: which works have proposed the operation but no solution or an informal solution is given; which works provide a formalisation of these operations; and which tools provide a support for them. To this list of operation we have added a new operation that we call unique cardinalities, a new operation that has arisen as a side result of this dissertation.

2.6 CONSTRAINT SATISFACTION PROBLEMS

Constraint programming is a mature field in artificial intelligence [58]. The expressive power of *Constraint Satisfaction Problems (CSPs)* and the wide catalogue of open-source solvers has turned them into the most used paradigm to deal with the AAFM. Constraint programming is a constantly evolving research topic where many

Relationship explanation operations			
Operation	**Informal or no semantics**	**Formal semantics**	**Tool**
	Why not?		
Invalid configuration	[5, 53, 81, 85, 96]	[93]	[6, 90]
Void FM	[5, 53, 85]	[81, 89, 93, 100]	[6, 90]
Dead Feature	[85, 89]	[89, 93]	[6, 90]
False-optional	[85, 89]	[89]	[90]
Optimisation	[85]	-	[90]
Wrong cardinalities	[85]	-	[90]
Unique cardinalities	-	-	-
	Why?		
Core features	[85]	-	-
Variant features	[85]	-	-
Valid configuration	[85]	-	-
Optimisation	[85]	-	-

Table 2.2: Current support for relationship explanations

algorithms and heuristics are arising yearly to improve the performance of commercial and free solvers.

A CSP is a declarative paradigm to model and solve problems using *constraints* [92]. It is defined as a 3–tuple (V, D, C) where V is a set of variables, each ranging on a finite domain from set D, and C is a set of constraints restricting the values that the variables can take simultaneously. A solution to a CSP is an assignment to each variable of a value from its corresponding domain so that all constraints are satisfied simultaneously.

Consider for instance, the CSP: $(\{a,b\}, \{\ \{0,1,2\}, \{0,1,2\}\ \}, \{a+b<4\})$ where both variables a and b take value in the domain $\{0,1,2\}$ and are constrained by $\{a+b<4\}$. The only value assignment that does not satisfy $a+b<4$ is $\{a \mapsto 2, b \mapsto 2\}$, so there are eight solutions.

There are four basic operations to obtain conclusions from CSPs:

Operation 1 - Searching for all the solutions.
This operation searches for all the value assignments that satisfy all the constraints in

the CSP. For the given example, this operation obtains 8 solutions.

Operation 2 - Searching for one random solution.
This operation searches for one solution that satisfies all the constraints. It is a non-deterministic operation since the solution obtained depends on the search algorithms which can use random criterion or heuristics to search for a solution.

Operation 3 - Satisfiability.
A CSP is satisfiable if there is at least one valid value assignment. This operation detects if a CSP is valid or not. An example of inconsistent CSP take is $(\{a,b\},\{\ \{0,1,2\},\ \{0,1,2\}\,\{a+b<0\})$ since there is no possible value assignment satisfying the constraints.

Operation 4 - Constraint propagation.
Searching for an equivalent CSP such that the available values in the domain are reduced according to the constraints in the CSP. For example, this operation obtains for an input CSP $(\{a,b\},\{\ \{0,1,2\},\ \{0,1,2\}\,\{a+b<2\})$, another equivalent CSP in the form $(\{a,b\},\{\ \{0,1\},\ \{0,1\}\,\{a+b<2\})$ since it is not possible to find for any solution that assigns 2 as a value for either a or b.

In many real–life applications, it is needed to find a good solution to a CSP rather than anyone. A solution quality or goodness is usually measured by an application–dependent function called *objective function*. The goal is finding a solution that satisfies all the constraints and minimise or maximise the objective function. Such problem is known as a *Constraint Optimisation Problem (COP)* is a 4-tuple (V,D,C,O) that adds an optimisation function O to a CSP. An objective function maps every solution in the CSP to a numerical value that is used for maximisation or minimisation. The adoption of COPs allow to define a new operation:

Operation 5 - Optimisation.
This operation searches for the solution/s that satisfy all the constraints in the CSP and minimise or maximise the objective function. If we define a COP $(\{a,b\},\{\ \{0,1,2\},\ \{0,1,2\}\ \},\{a+b<4\})$ where the optimisation function is $O(s)=a$, which maximises the value of a. There are two solutions in the original CSP $\{\ \{a\mapsto 2, b\mapsto 0\},\ \{a\mapsto 2, b\mapsto 1\}\ \}$, that maximises the value of the objective function and are therefore the solutions of the COP.

2.7 SUMMARY

This Chapter presents the main concepts on which this dissertation relies. FMs and configurations have been presented as the two models used in SPLs to represent all the products that can be built and the user decisions to find for the most suitable product among the available ones.

The AAFM is introduced as an approach to extract information from FMs and configurations. Some examples of AAFM operations have been presented and the general schema used to solve them. Specifically, we focus on explanatory operations, detailing their state of the art.

The AAFM usually relies on declarative techniques to support the analysis. The most used approach are CSPs. They have been briefly introduced to show their declarative capabilities.

In the next Chapter we introduce a declarative framework to represent problems from which information can be extracted using deductive and abductive reasoning.

3

Deduction and Abduction Problems

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution. It is, strictly speaking, a real factor in scientific research

Albert Einstein (1879 – 1955),
Scientist

The main approach to deal with the AAFM is transforming a FM into a declarative paradigm where the analysis operations are interpreted in terms of operations in that paradigm. Despite of the differences among those paradigms, they share the way of reasoning that they implement. In this dissertation we conjecture that most of the analysis operations in the AAFM can be interpreted in terms of deduction and abduction, two well-known ways of reasoning about a given knowledge. In this Chapter we present the Deductive and Abductive Problem (DAP) framework to describe problems in such terms that deduction and abduction can be used to extract information from them by means of operations. In Section §3.1 we propose Deduction Problems (DPs) and define the operations that are available to perform deduction. In Section §3.2 we propose Abduction Problems (APs) and define the operations that are available to perform abduction. In Section §3.3 we summarise the contributions of this Chapter and advance how DAPs are used in this dissertation.

3.1 DEDUCTION PROBLEMS

Deductive reasoning or deduction is a form of reasoning that permits to obtain conclusions from a previous knowledge. It is a way to make explicit an information that is implicit in a knowledge representation. In the AAFM, many different paradigms such as *First Order Logic (FOL)*, CSPs, SAT or BDD have been used to perform operations like validation, errors existence or products counting. These paradigms use deduction to extract implicit information in FMs.

We aim to generalise existing proposals in such a form that will be as independent as possible from a particular paradigm. For that sake, we propose a declarative framework to represent problems that aim to be analysed in terms of deduction that we coin as *Deduction Problem (DP)*. A DP is a model of a real-world scenario in a form that can be used for deductive reasoning. It relies on FOL[1] to describe the knowledge but we envision that many other logics might fit into our approach. A DP is defined as follows:

Definition 3.1 - Deduction problem.
A DP is a 4-tuple $DP(C,V,K,S)$ where C stands for constants, V for variables, K for a set of formulas that comprise the *Knowledge Base (KB)* and S for predicates semantics. Constants and variables denote real-world elements. The semantics defines the kinds of predicates that can be used to describe the problem and their meaning. The KB relies on these kinds of properties in the semantics to describe the specific relationships among real-world elements.

Figure 3.1: A one-inverter circuit

As an example of a DP, let us take the one-inverter circuit in Figure §3.1. In digital circuits theory, boolean information is represented as two levels of voltages. Usually 0 is represented by 0V and 1 by +5V. An inverter behaves outputting 1/+5V for a 0/0V

[1]Besides basic elements in FOL (see Appendix §C) we use equality and assume the definition of arithmetic operations wherever they are used.

input and 0/0V for a 1/+5V input. A DP for the one-inverter circuit can be described as follows:

$$\begin{aligned} C &= \{I_1, 0, 1\} \\ V &= \{i_1, o_1\} \\ KB &= \{inverter(I_1), in(I_1, i_1), out(I_1, o_1)\} \\ S &= \{inverter(i) \equiv \exists x \cdot x \in \{0,1\} \wedge in(i,x) \wedge out(i, 1-x)\} \end{aligned}$$

The constants represent the elements in the one-inverter circuit: I_1 for the inverter; 0 and 1 for the two boolean values that can be given in a digital circuit. The variables represent the unspecified elements in the circuit, in this case i_1 for the input and o_1 for the output. The semantics defines the meaning of predicates that can be used in the KB. In this case the meaning of $inverter(i)$ predicate is given by the semantics: there exist an x variable whose value is between 0 and 1 such that x is the input of the inverter and $1 - x$ is its output. In the KB the *inverter* predicate is used to bind the constant I_1 to the inverter in the circuit. *in* and *out* predicates are used to bind i_1 and o_1 variables to the input and output of the inverter respectively.

The relationship among constant and variables and real-world elements can be stored in a *traceability table*. This table is used to trace the conclusions obtained from the DP to real-world elements. Table §3.1 shows the traceability table for the inverter circuit.

Traceability table	
Constants	
I_1	an inverter
0	0V input
1	+5V input
Variables	
i_1	the input of the inverter
o_1	the output of the inverter

Table 3.1: Traceability table for the one-inverter circuit DP

In order to extract information from a DP, we propose three *deductive operations*:

Operation 1 - Solutions.
This operation finds all the possible solutions for a DP, being a solution an assignment of constants to variables that holds with the KB. We denote this operation in the following form:

$$solutions : DP \rightarrow S$$

Such that S is a set of solutions, being a solution the cartesian product $V \times C$ that assigns a constant for each variable in the DP. For the one-inverter circuit example, let δ be the DP that describes it. $solutions(\delta) = \{a_1, a_2\}$, where $a_1 = \{i_1 \mapsto 1, o_1 \mapsto 0\}$ and $a_2 = \{i_1 \mapsto 0, o_1 \mapsto 1\}$. They correspond to the valid inputs and outputs in the circuit. Using the traceability table to interpret the meaning of a solution in terms of the problem, the assignment a_1 indicates that whenever the inverter is given a 0V voltage as an input, a +5V voltage is obtained as an output.

Operation 2 - Satisfiability.
This operation determines if a DP is satisfiable, i.e. if there is at least one assignment such that all the formulas in the KB hold. We denote this deductive operation as a function in the following form:

$$isSatisfiable : DP \rightarrow \{true, false\}$$

Let δ be the DP for the circuit example, then $isSatisfiable(\delta) = true$ since there are two valid assignments that can be obtained with the solutions operation. This operation can be defined in terms of model finding as follows:

$$isSatisfiable(\delta) \equiv solutions(\delta) \neq \emptyset$$

Despite of the possibility of defining this operation in terms of solutions operation, we prefer to keep this operation separately since the deduction techniques used to solve this operation usually differ from the techniques used for obtaining all the solutions.

Operation 3 - Inference.
This operation determines if it is possible to infer a given conclusion γ from a DP. In logical terms, γ is a logical consequence of a KB if $KB \models \gamma$ [2]. We denote this deductive operation as a function in the following form:

[2] $a \models b$ represents that b is a logical consequence of a for the given semantics in S. In the proposed examples, it can be interpreted as an entailment for a FOL

$$isInferred : DP \times A \rightarrow \{true, false\}$$

such that A is the set of all the γ formulas that can be formed from the syntax described by the DP. For the circuit example, if we want to check if the input for the inverter can be either 0V or +5V, we define a formula $\gamma = in(I_1,0) \vee in(I_1,1)$ that describes this potential conclusion. Let δ be the DP for the circuit example, then $isInferred(\delta,\gamma) = true$ so the formula can be logically concluded from the DP.

Although we are only using these three deductive operations in this dissertation, we also propose a fourth operation related with model checking in FOL. This operation could be useful in other contexts where DP would be applicable. We define solution checking operation as follows:

Operation 4 - Solution Checking.
This operation checks if a certain assignment of constants to variables is a solution for the DP, i.e. the assignment holds with the KB. We denote this operation as a function in the following form:

$$isSolution : DP \times (V \times C) \rightarrow \{true, false\}$$

For the circuit example, let δ be the DP that describes it, $a_1 = \{i_1 \mapsto 0, o_1 \mapsto 1\}$ an assignment for a 0V input and +5V output, and $a_2 = \{i_1 \mapsto 0, o_1 \mapsto 0\}$ an assignment for 0V for input and output. a_1 is a solution since $isSolution(\delta,a_1) = true$, while a_2 is not a solution since $isSolution(\delta,a_2) = false$.

We use DPs as an abstraction that allows us to provide a solution for certain analysis operations without binding to a specific implementation paradigm. Despite there is an absence of off-the-shelf solvers for DPs, they can be solved by means of existing solvers recurring to transformations. As shown in Appendix §B there exist in the literature many works that propose the transformation from FOL to other implementation paradigms. Since DPs are clearly inspired in FOL, we devise the possibility of defining transformations from DPs to other paradims where solvers could be used to carry out deductive reasoning.

3.2 ABDUCTION PROBLEMS

Abductive reasoning or simply abduction, is used to obtain probable explanations why certain behaviours are given from a set of probable hypotheses. For example if we want to know the reasons why a car does not start, abduction allows us to contemplate that the car has run out of battery or the petrol tank is empty. Both explanations could be valid at the same time, although it is less probable than only one of them. To determine the accurate failure in the car, new information or *observations* is needed in order to increase the certainty about explanations. For example, watching if the petrol tank is empty or not contributes to discard or confirm the previous explanations.

As we propose for deduction, we aim to generalise existing abduction proposals in such a form that will be as independent as possible from a particular paradigm. For that sake, we propose a declarative framework, inspired by the Theorist framework [71], to represent problems that aim to be analysed in terms of abduction that we coin as *Abduction Problem (AP)*. An AP is a model of a real-world scenario in a form that can be used for abductive reasoning. An AP is defined as follows:

Definition 3.2 - Abduction Problem.
An AP is a 5-tuple $AP(C,V,(F,H,O),S,min(X))$ where C stands for constants, V for variables, (F,H,O) for the KB that is structured in three subsets of formulas, S for the predicates semantics, and $min(X)$ for minimality criterion. The constants (C) and variables (V) sets and the semantics (S) represent the same concepts than in a DP. The 3-tuple (F,H,O) represents the *facts* (F) that are known to be true, the observed behaviour or *observation* (O) and the set of *hypotheses* (H) that can be used to explain the observations. An AP must satisfy the precondition $F \not\models O$, i.e. the observation cannot be concluded from the set of facts. An explanation is a subset of hypotheses such that it explains the given observation together with the set of facts. So $\Delta \subseteq H$ is an explanation iff

$$F \not\models O \text{ and } F \cup \Delta \models O$$

The minimality criterion ($min(X)$) allows to reduce the set of explanations to those that are minimal according to this criterion.

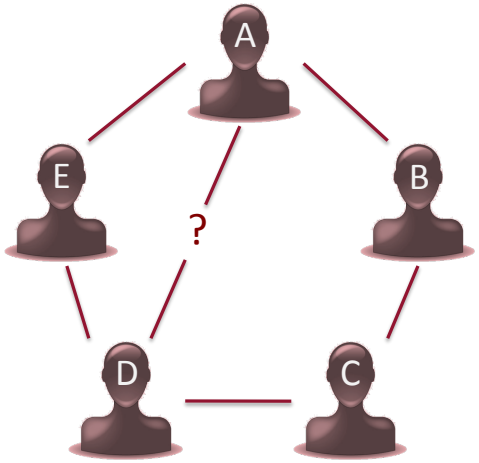


Figure 3.2: The five-person example

Following we describe an example of an AP that models the friendship among five random persons A, B, C, D and E as shown in Figure §3.2. Some of these persons are mutual friends. Those who are not friends can be introduced by common friends and they will become friends. Friends A and D are not mutual friends initially. In this example we want to obtain the best way to introduce A and D, which is a case of abduction. For that sake, we define the following AP that describes this problem:

$$
\begin{aligned}
C &= \{\ A,B,C,D,E\} \\
V &= \emptyset \\
F &= \{\ friend(A,B), friend(B,C), friend(C,D), friend(D,E), friend(E,A), \\
&\quad\ friend(x,y) \Leftrightarrow friend(y,x)\} \\
H &= \{\ introduce(A,B,C), introduce(A,B,D), ..., introduce(C,D,E)\} \\
O &= \{\ friend(A,D)\} \\
S &= \{\ introduce(x,y,z) \equiv friend(x,y) \wedge friend(y,z) \Rightarrow friend(x,z)\}
\end{aligned}
$$

The constants (C) represent each person. The semantics (S) defines how a new friendship arises when two persons are introduced by a common friend. The set of facts describes the initial friendship among the five persons and the transitivity of friendship. All the possible introductions are placed in the set of hypothesis (H). Turning A and D into friends is the observation or goal of the AP. As a prerequisite of any AP, the observation cannot be inferred from the set of facts, as it is the case since A and D do not know each other a priori. An explanation for this problem is a set of *introduce* predicates taken from the hypotheses set that can be used so that A and D can be friends. So for example $\{introduce(A,E,D)\}$ is an explanation for the AP since

$$friend(E,A), friend(D,E), introduce(A,E,D) \models friend(A,D)$$

which means that A and E are common friends of D and they can be introduced to turn A and D into friends.

3.2.1 Minimal explanations

In the friendship example, there are 120 possible explanations, but we are not interested in all of them, but in those that are more probable. According to the Occam's razor principle or parsimony law, the succinctest or minimal explanations are the most probable ones. Minimal explanations are usually sufficient to explain a situation and in case they are not, they guide the process to obtain new observations that delimit the explanations. Different minimality criteria can be used in to obtain minimal explanations:

- Minimal subset: an explanation is minimal if no proper subset of it is also an explanation:
$$min_S(E) \equiv \{e | e \in E \wedge \neg \exists e' (e' \in E \wedge e' \subset e)\}$$
- Minimal number of assumptions: an explanation is minimal if there exists no explanation with a fewer number of elements.
$$min_{\#A}(E) \equiv \{e | e \in E \wedge \neg \exists e' (e' \in E \wedge |e'| < |e|)\}$$
For example the minimal explanations obtained for the friendship problem can be reduce to $\{introduce(A,E,D)\}$ since it only has one element while the other two explanations have two elements.
- Minimal weighted explanation: weights are defined for each formula in the hypotheses set, establishing an order relationship $\sqsubseteq_p$ between hypotheses. An explanation is minimal if there exists no other explanation with less weight:
$$min_\omega(E) \equiv \{e | e \in E \wedge \neg \exists e' (e' \in E \wedge e' \sqsubseteq_p e)\}$$
- Circumscription: same as minimal number of assumptions criterion but only a kind of predicates from the hypotheses set is taken into account ignoring remaining ones. For example, in the friendship AP only *introduce* predicates can be taken into account for explanations.

Given the above minimality criteria, we propose an abductive operation to perform minimal explanations on APs:

Operation 1 - Minimal Explanation.
This operation obtains the minimal explanations for an AP. We denote this operation as a function in the following form:

$$minExpl : AP \rightarrow H^*$$
$$minExpl(AP(C,V,(F,H,O),S,min(X))) \equiv min(\{\Delta | \Delta \subseteq H, F \not\models O, F \cup \Delta \models O\})$$

For the friendship example, let α be the AP that represents the problem. If the minimal subset criterion is used, the following minimal explanations are obtained:

$$\begin{aligned} minExpl(\alpha) = \{ \quad & \{introduce(A,E,D)\}, \\ & \{introduce(A,B,C), introduce(A,C,D)\}, \\ & \{introduce(B,C,D), introduce(A,B,D)\}\} \end{aligned}$$

3.2.2 Conflict sets

When a DP defines a KB with contradictory formulas, the DP is said to be not satisfiable, i.e. it has no solutions. In this situation it is important to obtain the set of formulas in the KB that are in conflict. Obtaining such sets can be interpreted in terms of abduction. For that sake, the KB is divided in two sets of facts and hypotheses, i.e. $KB = F \cup H$. The set of facts contains the set of formulas in the KB that are certainly known to be correct, if any. The set of hypotheses contains all the remaining formulas that can be provoking the unsatisfiability. These two sets are used to define an AP in the form $AP(C,V,(F,H,\emptyset),S,min(X))$.

Since the set of observations is empty, there is no behaviour to explain, so the minimal explanation operation cannot be used to detect contradicting formulas. In this situation, we are interested in searching for the minimal conflict sets, which are defined as follows:

Definition 3.3 - Conflict Set.
Let α be an AP in the form $(C,V,(F,H,\emptyset),S,min(X))$ such that the set of facts F is consistent. A subset of hypotheses $\Delta \subseteq H$ is said to be a conflict set iff Δ contradicts the knowledge in the set of facts, i.e. $F \cup \Delta \models \perp$

Let us consider a catalogue of cars as an example. A car has three properties: transmission, colour and cost. There are three available cars with different properties. A customer is searching for a black car with a manual transmission that costs less than 15,000€. However, it is not possible to find a car with such properties. In this case, we build an AP that distinguishes among facts and hypotheses. The set of facts contains the database of cars, since the catalogue cannot be changed. The set of hypotheses contains the customer decisions, which can be relaxed or changed to find for a suitable car. The AP that describes this problem results as follows:

$$
\begin{aligned}
C = \; & \{C_1, C_2, C_3, GEAR, MANUAL, AUTO, COLOR, RED, BLACK\} \cup \mathbb{N} \\
V = \; & \emptyset \\
F = \; & \{car(C_1, MANUAL, RED, 12000), \\
& car(C_2, AUTO, RED, 13500), \\
& car(C_3, MANUAL, BLACK, 16000), \\
& value(GEAR, MANUAL) \oplus value(GEAR, AUTO), \\
& value(COLOR, BLACK) \oplus value(COLOR, RED)\} \\
H = \; & \{value(GEAR, MANUAL), \\
& \exists x \cdot value(COST, x) \wedge x < 15000, \\
& value(COLOR, BLACK)\} \\
O = \; & \emptyset \\
S = \; & \{car(model, gear, color, cost) \equiv product(model) \Leftrightarrow value(GEAR, gear) \wedge \\
& value(COLOR, color) \wedge value(COST, cost)\} \\
min(X) = \; & \{x | x \in X \wedge \neg \exists x' (x' \in X \wedge x' \subset x)\}
\end{aligned}
$$

There is no car satisfying the customer's demand because $DP(C, V, F \cup H, S)$ is not satisfiable. Let us name the three hypotheses as $H_1 = \{value(GEAR, MANUAL)\}$, $H_2 = \{\exists x \cdot value(COST, x) \wedge x < 15000\}$ and $H_3 = \{value(COLOR, BLACK)\}$. The conflict set operation detects six conflict sets: $\{H_2\}$, $\{H_3\}$, $\{H_1, H_2\}$, $\{H_1, H_3\}$, $\{H_2, H_3\}$ and $\{H_1, H_2, H_3\}$, As for minimal explanations, we are not interested in all the potential conflict sets, but in those sets that are minimal. Therefore we define the following operation for this purpose:

Operation 2 - Minimal Conflict Sets.

This operation obtains the set of minimal conflict sets for a given AP. We denote this operation as a function in the following form:

$$
\begin{aligned}
& minConflicts : AP \rightarrow H^* \\
& minConflicts(AP(C, V, (F, H, \emptyset), S, min(X))) \equiv min(\{\Delta | \Delta \subseteq H, \quad F \text{ is consistent} \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad F \cup \Delta \models \perp\})
\end{aligned}
$$

For the above example and given the minimal subset criterion, the minimal conflict sets are $\{H_2\}$ and $\{H_3\}$ since remaining conflict sets are supersets of them. So the customer must resign either the black colour or the cost restriction.

3.3 SUMMARY

In this Chapter we propose DPs and APs as two frameworks that support deductive and abductive reasoning, two forms of reasoning used in the AAFM. These frameworks aim to make reasoning independent of specific paradigms or solvers, offering an abstract layer where problems can be defined in general terms and reasoning can be performed by means of operations.

One of the problems we have found in the AAFM is the myriad of declarative approaches that solve the same operations in terms of different paradigms such as CSPs, *Binary Decision Diagrams (BDDs)* or SAT. In [85] we had the intuition that most of the AAFM operations could be interpreted independently of the specific paradigm as a form of deduction and abduction. That work has led us to propose DPs and APs as frameworks to describe problems that use both forms of reasoning. In this dissertation, we describe SFMs in terms of DPs in Chapter §7 to perform analysis operations that rely on deduction, and in terms of APs in Chapter §8 to perform analysis operations that rely on abduction. This way, DPs and APs enable to give an operational semantics to analysis operations in a solver-independent form.

Part II

Our contribution

4

Motivation

The people who get on in this world are the people who get up and look for the circumstances they want, and, if they can't find them, make them.

George Bernard Shaw (1856–1950),
Playwrighter and Nobel Prize in Literature

Once we have raised a number of research questions and goals, we aim to motivate the problems and to justify the solutions that we approach in this dissertation. In Section §4.1, we analyse current proposals on configuration modelling, explanatory analysis and formalisation, which lead to state three research questions and to define goals to answer them. In Section §4.2 we discuss our proposal to achieve these goals. Finally, Section §4.3 summarises the conclusions and justifies the following structure of our contribution.

4.1 Analysis of current solutions

This dissertation aims to overcome the drawbacks that we have found in AAFM regarding modelling and analysis. This Section focuses on remarking the limitations that have led to state the research questions in Chapter §1. First, we study why current FMs are not fully-configurable due to the expressiveness of CMs in Section §4.1.1. Next, we present the limitations we have found in explanatory analysis in Section §4.1.2. Finally, we analyse the AAFM formalisation challenges in Section §4.1.3.

4.1.1 Configuration models

CMs have not been specifically studied in the AAFM, being widely accepted the three-set model that represents user decisions in terms of selected, removed and undecided features. We have found three limitations in this CM that we aim to save in this dissertation.

First, despite cardinalities and attributes are used in ECBFMs to represent relevant information, current CMs are unable to represent decisions on them, only enabling the decision making on features. It impedes a user making decisions about attributes such as 'I want a SHS that costs less than 2.000 €', or about cardinalities such as 'I want a SHS with two Internet connections'. We affirm that current FMs are not *fully-configurable* since CMs are unable to make decisions on any element of the FM.

Second, a strong relationship between elements in the FM and in the CM arises when attributes and cardinalities come into play. A CM must ensure that a user decision on an attribute is made within the attribute domain, which is specified in the FM. The same happens for cardinalities, whose valid values for user decisions are modelled in the FM. FMs and CMs share elements and the values those elements can take in a user decision. There are two main approaches to deal with information sharing: keeping two separate models in constant synchronisation, communicating any change in the FM to CMs, or combining FMs and CMs in a unique model around the common elements.

Third, some authors [20, 99] have used annotations on FMs to depict decisions on features. But there is no graphical representation that supports attributes and cardinalities.

From these problems we have found regarding CMs, we state the following re-

search question:

Is it possible to extend CMs to enable fully-configurable ECBFMs?

In order to find an answer for this question, we define three goals:

1. Propose a CM that enables the representation of user decisions on attributes and cardinalities.
2. Taking advantage of the information that FMs and CMs share.
3. Representing CMs together with FMs in a backwards-compatible graphical notation.

4.1.2 Explanatory analysis

The explanatory analysis covers configuration and relationships explanation, where the research has progressed unevenly. Regarding configuration explanations, the work of White et al. [99] can be considered as the most complete one to date since they are able to explain and repair configurations in parallel configuration processes. However, they are only able to deal with user decisions on features, ignoring attributes and cardinalities. This is a common limitation of every studied proposal.

Regarding relationships explanations, we have detected several open issues in the works we have analysed. First, a formal semantics has only been given to a subset of operations [53, 81, 85, 96]. In particular those operations in which cardinalities and attributes are involved such as wrong and unique cardinalities, and optimisation, have been left out of any proposal. Second, the tools support more operations than those that have been published to date. There are operations that have been implemented but no formal semantics has been given, making it impossible to verify their correctness or to replicate them in other tools.

According to this scenario, we have found three main limitations in the explanatory analysis that we aim to save in this dissertation. First, we can affirm that current proposals on the explanatory analysis are not available for fully-configurable FMs and therefore they are unable to deal with cardinalities and/or attributes.

Second, we have detected that the implicit procedure to solve most of the relationship explanations consists of simulating a user setting a configuration and then performing an explanation of a valid or invalid configuration. We conjecture that those

operations that have not been given a semantics is due to the inability to simulate user decisions about cardinalities and attributes. Thus, enabling fully-configurable FMs would allow to make decisions on cardinalities and attributes and therefore we can give a formal semantics to these operations.

Third, the research on explanatory analysis is clearly separated in those works that focus on configuration explanation or on relationships explanation, existing no proposal of a unified solution. However, there is an implicit connection between both kinds of operations in the techniques used to solve them. White et al. [99] remarks that their work in configuration explanation is inspired in the interpretation made by Trinidad et al. [89] of relationships explanation. It led us to intuit in [85] an interpretation of the explanatory analysis as a form of abduction.

From these problems we have found regarding the explanatory analysis, we state the following research question:

Is it possible to improve the current support for the explanatory operations on ECBFMs?

In order to find an answer for this question, we define three goals:

1. Applying explanatory operations to fully-configurable ECBFMs.
2. Identifying and formally define the comprehensive set of undiscovered explanatory operations.
3. Interpreting explanatory operations as abduction problems.

4.1.3 Formalisation of the AAFM

In general, the use of formal approaches in the AAFM literature is very extended. Most of the works rely on declarative paradigms that can be executed by existing solvers, which confer them an operational semantics. First works proposed solutions for a subset of basic AAFM operations in terms of CSPs and Prolog [10, 53, 59]. As new operations were proposed such as metrics or error detection [87] that use basic algorithms on top of CSP solvers, the declarative nature of AAFM starts to be compromised.

With the coming of explanations, the mapping to CSP used to date must be modified, introducing new artificial variables and an optimisation function, loosing even

more the declarative nature of the proposal. This is the reason why Trinidad et al. [89] interpreted the explanatory analysis in terms of theory of diagnosis, to stick to the general schema of the AAFM.

Over time, the proposed solutions have lost the style offered by declarative programming and logic solvers. Perhaps for this reason, there have been numerous studies that suggest to interpret the same operations that have already been solved previously using different paradigms, justified by improving response times in experiments which most of the times are not properly characterised. According to Benavides et al. [14], 35 papers propose solutions for the void FM operation and 16 papers for counting the number of products. However, it is unclear the added value that is offered with respect to each other. From all the studies analysed, none offers a unique formal approach to the AAFM that covers most of the current analysis operations.

We have found three open issues in the AAFM. First, the impact of fully-configurable ECBFMs in the existing analysis operations claim for a revision of the catalogue of AAFM operations to support them. Second, the number of operations in the AAFM is increasing but new operations can be given a semantics in terms of other analysis operations. Using this approach would significantly reduce the formalisation and tooling efforts. Third, there is no formalisation approach that deals with the AAFM in unified terms and focusing on the common points among operations rather than proposing operation-specific solutions.

According to this scenario, we state the following research question:

Is it possible to improve current formal specification frameworks for the AAFM?

In order to find an answer for this question, we define three goals:

1. Defining a new, comprehensive analysis operations catalogue encompassing explanatory and non-explanatory operations on fully-configurable ECBFMs.

2. Identifying a minimum set of basic operations (cannot be defined in terms of other operations) on the basis of which the remaining analysis operations in the catalogue can be defined.

3. Interpreting non-explanatory operations as deduction problems, and therefore stating all the analysis operations as DAPs.

4.2 Discussion

This dissertation aims to save the above limitations, contributing to improve SPL modelling and the automated analysis.

First, adding a support for fully-configurable FMs is not as easy as adding attributes and cardinalities to the three sets of selected, removed and undecided elements. That approach works for features but not for cardinalities and attributes. For example if a cardinal is removed in a cardinality, such as $\{3\}$ in $[1..3]$, then the cardinality is not undecided anymore, but neither $\{1\}$ nor $\{2\}$ are selected values yet. And also for attributes whose range of values can be even wider and continuous such as an Internet bandwidth from 1 to 100 Mbps. Thus, the three-set configuration model is not suitable for cardinalities and attributes that must represent the values that are still available for future decisions, therefore a new kind of model has to be proposed.

We take advantage of the definition of SFMs to save other limitations of current CMs. First, SFMs must support multiple users interacting with the model at the same time in parallel. Second, it is frequent that in order to perform certain analysis operations, they simulate an user making a decisions. So for example dead feature detection simulates automatically a user selecting the feature under evaluation and checks if the resulting configuration is valid. The model must distinguish these automatic decisions from user decisions. Third, when FMs and CMs are used jointly in analysis operations, the first task to perform is to check that there exist no element in the configuration that is missing in the FM. SFMs must guarantee that point by construction. Last, a graphical representation of SFMs is desirable, which explicitly depicts user decisions.

The introduction of SFMs as fully-configurable FMs forces us to thoroughly revise the current configuration explanation proposals to support decisions on cardinalities and attributes. Moreover, we conjecture that SFMs enable the formalisation of all the relationship explanation operations proposed up to date and others that have not been still proposed. In order to formalise both kinds of explanatory operations, we propose the interpretation of them as APs.

But the impact of SFMs in the automated analysis in not only limited to explanations. Many non-explanatory operations also use CMs as inputs so they must be revised to incorporate SFMs instead. We propose a formalisation of these operations as DPs.

Interpreting the automated analysis as a DAP enables the simplification of the cur-

rent catalogue of analysis operations. We can find a correspondence between a subset of analysis operations and DAP operations. Remaining operations combine them together with some model manipulation operations. Due to this correspondence, we suggest a simplified catalogue of analysis operations that distinguishes between basic and compound operations.

Simplifying the catalogue enables the construction of fully-featured implementations of analysis engines for SFMs with a minimum set of analysis operations.

Building tools is a mandatory task in software engineering research to demonstrate the feasibility of our ideas. For that reason, there is a need to build a prototype tool that makes benefits from the simplification of the catalogue.

4.3 SUMMARY

This Chapter briefly introduces the limitations that we have found in the AAFM regarding modelling and analysis. We have stated three research questions that have defined the goals of this dissertation. First, we claim that current FMs are not fully-configurable, which leads to important limitation in the AAFM. Second, the explanatory analysis still has open issues in the form of analysis operations that still remain unsolved. We claim that the adoption of fully-configurable FMs and the interpretation of explanations as APs would enable the formalisation of these unsolved operations. Third, using fully-configurable FMs implies the revision of all the catalogue of more than 30 AAFM operations. These operations must be given a formal semantics.

We discuss how we conjecture that these limitations can be saved, proposing a solution for all the posed research questions. We firstly propose to create SFMs as a new kind of fully-configurable model that unifies FMs and CMs, Then a revision of existing explanatory operations is demanded, proposing their interpretation as APs. The introduction of SFMs also affects to non-explanatory operations, what forces us to revise the existing formalisation. We propose interpreting them as DPs. Last, we propose a simplification of the catalogue of analysis operations, defining a set of basic operations on top of which remaining analysis operations can be defined.

The following Chapters present our contributions. First, SFMs are presented in Chapter §5. Then, we propose the AASFM as a new analysis paradigm in Chapter §6. Chapter §7 presents an interpretation of non-explanatory or query operations as DPs, Chapter §8 presents an interpretation of explanatory operations as APs. Chapter §9

gives a semantics for most of the existing analysis operations as a composition of the basic operations defined in previous Chapters. Last, Chapter §10 describes the tool that we have built to demonstrate the feasibility and correctness of our proposal.

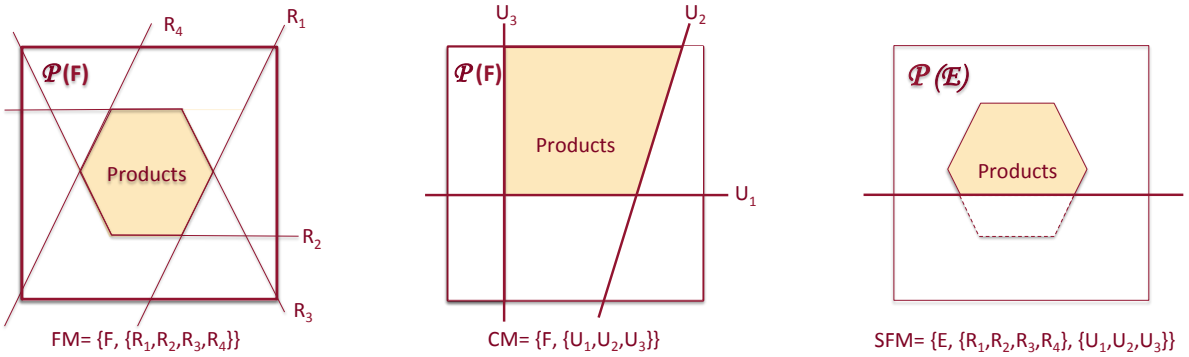
5

Stateful Feature Models

We have no idea about the 'real' nature of things... The function of modelling is to arrive at descriptions which are useful.

Richard Bandler and John Grinder,
Psychologists

Stateful feature models (SFMs) are our proposal to unify FMs and CMs in a single model. In Section §5.1, FMs and CMs are interpreted as a set of constraints on the same set of elements. In Section §5.2 we describe the main concepts in SFMs, and in Section §5.3 we propose an abstract model for SFMs. We propose two different representations of SFMs each of them for a different purpose. We firstly present the stateful feature diagrams as a graphical representation of SFMs in Section §5.4. Second, a UML stateful feature metamodel is proposed in Section §5.5 that enables the transformation from SFMs into other metamodels. Last, Section §5.6 presents a summary of this Chapter.



(a) Feature model (b) Configuration model (c) Stateful feature model

Figure 5.1: A visual metaphore of FMs, CMs and SFMs

5.1 RATIONALE

A FM describes the set of products in terms of features and relationships. A product is a subset of features that satisfy all the relationships. From these definitions, we can interpret a FM in terms of the following sets. Let F be the set of features in a SPL, and let $\mathcal{P}(F)$ be the set of all the possible combination of features. The relationships can be interpreted as constraints on $\mathcal{P}(F)$. These constraints define the set of products (P) as a subset $P \subseteq \mathcal{P}(F)$.

CMs represent the user decisions in terms of three sets of selected, removed and undecided features. User decisions can be interpreted as another different way to constrain the set of all the possible combination of features $\mathcal{P}(F)$ with a different kind of constraints. Therefore FMs and CMs can be regarded as two different models that describe subsets of $\mathcal{P}(F)$.

This vision also fits into CBFMs and EFMs where cardinalities and attributes are added to the set of features. In this case, a new set that represents all the features, cardinalities and attributes in an ECBFM is necessary. This set is denoted as the set of elements E. In this case, relationships can be interpreted as constraints on $\mathcal{P}(E)$, i.e. the space of all the possible combinations of elements. Symmetrically, user decisions in CMs could also be interpreted as constraint on $\mathcal{P}(E)$ if they were fully-configurable, but current CMs keep on interpreting the CM as a subset of $\mathcal{P}(F)$.

In order to enable fully-configurable FMs, we propose extending CMs to support cardinalities and attributes besides features. The resulting models must also satisfy three main requirements in order to save the limitations presented in Section §4.1:

(i) user decisions can only refer to elements in E so it must avoid any reference to invalid cardinalities or attribute values; (ii) CMs must allow multiple users making decisions at the same time, and (iii) CMs must distinguish between user and automatic decisions.

As a solution, we propose incorporating FMs and CMs together in a single model that we have coined as *Stateful Feature Model (SFM)*. The resulting model stores together the set of elements E, and two sets of constraints, one for relationships and another one for user decisions (see Figure §5.1). An SFM supports multiuser configurations and distinguishes between user and automatic decisions. SFMs are the first fully-configurable FMs, able to represent user decisions on attributes and cardinalities. Table §5.1 compares the capability of dealing with features, cardinalities and attributes in SFMs, basic FMs, CBFMs, EFMs and ECBFMs. In next Section we propose an abstract model for SFMs and the relevant concepts that arise from their use.

Elements affected by constraints†					
Constraint	**Basic FM**	**CBFM**	**EFM**	**ECBFM**	**SFM**
Relationship	F	F, C	F, A	F, C, A	F, C, A
Configuration	F	F	F	F	F, C, A

† F = Features, C = Cardinals, A = Attributes

Table 5.1: A comparison of the use of elements in different kinds of feature models

5.2 MAIN CONCEPTS

An SFM contains information about the elements in a SPL, which are features, cardinalities and attributes, hitherto stored in FMs. Depending on the decisions a user can make on elements, each element has its own set of states. So features have a selected or removed state; Cardinalities have as many states as cardinals in its range; Attributes have as many states as values they can take. An assignment of states for every element in the SFM defines the characteristics of what is called a *potential product*.

To determine which of the potential products can be built within the SPL, a set of *relationships* constrains the valid combinations of states, defining what is called the set of *SPL products*. An SFM also collects all user decisions, which are constraints that define a subset of potential products named *configuration*. The set of SPL products that

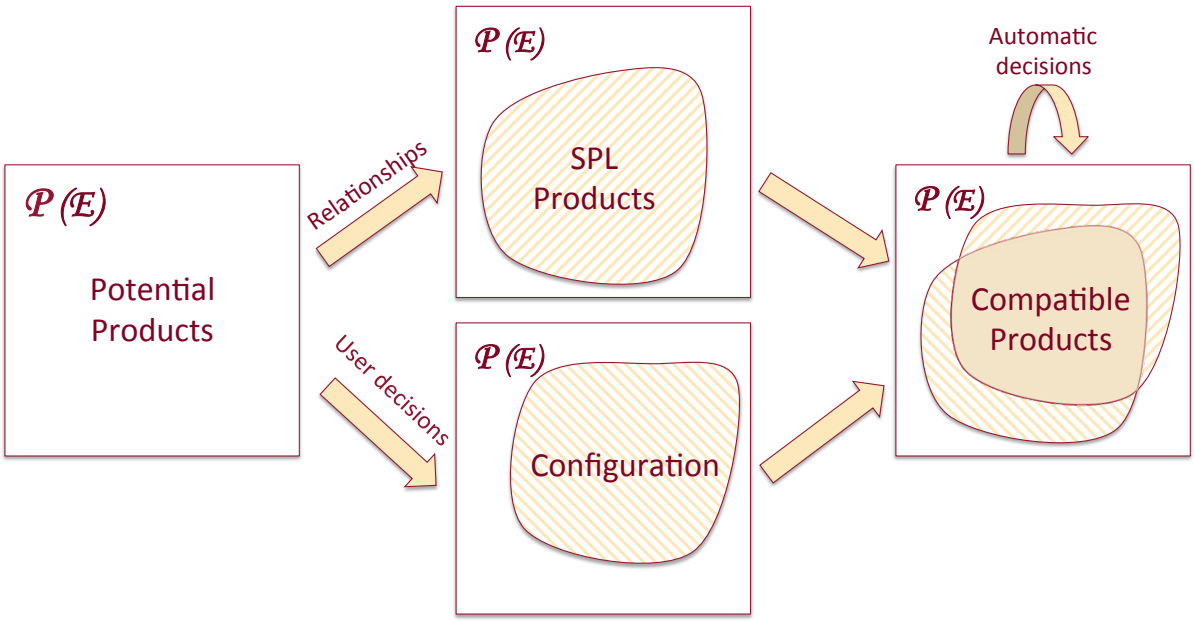


Figure 5.2: A geometrical interpretation of basic concepts in SFMs

are also found in the configuration is called the set of *compatible products*, and brings together the SPL products that meet all user decisions.

SFMs distinguish among user and automatic decisions. An automatic decisions is the one that is made by an AAFM operation to support the configuration process. An automatic decision can be defined as the one that if a user makes it, the set of compatible products remains unaltered. They are computed and stored separately from user decisions unless a user explicitly makes such a decision.

Figure §5.2 proposes a geometrical interpretation of the concepts presented in this Section.

5.3 Abstract model for SFMs

In this Section we give a transformational semantics to SFMs in terms of set theory, relying on a new vision of products as an assignment of states to elements. In FMs, each product is described as a different subset of features. In SFMs, all the products share the same set of elements, denoted by a non-empty set $E = \{E_1, ..., E_n\}$ where each E_i is an element in that model, either features, cardinalities or attributes. A product is defined as an assignment of states to every element in E such that each element

has a set of available states that depends on the kind of element. This way, features have selected or removed states to indicate their presence or absence in a product; a cardinality has a cardinal value as state to indicate the number of features that are in a selected state within a set relationship; attributes have a state which corresponds to a value that represents the quality or behaviour of a linked feature.

To define which states every element can have, an SFM has a set of available state sets $AS_1,...,AS_n$ such that AS_j is the set of available states for an element E_j. Let us consider an SPL with 5 elements $E = \{F_R, F_A, F_B, F_C, C_1, Mem\}$ as an example. The available states for each elements could be: $AS_{F_R} = \{sel\}$,$AS_{F_A} = AS_{F_B} = AS_{F_C} = \{sel, rem\}$,$AS_{C_1} = \{1,2,3\}$, $AS_{Mem} = \{128,256\}$. This way, the root feature F_R must have a selected state, features F_A, F_B and F_C can be either in a selected or removed state, C_1 cardinality, which affects features F_A, F_B and F_C, must be either 1, 2 or 3 and the product can have a 128 or 256 kilobytes memory size. We define the set of potential products as follows

Definition 5.1 - Set of potential products.
Let AS_j be the set of available states for the jth element $e_j \in E$. The set of all the potential products in an SFM is defined by the cartesian product $D = AS_1 \times ... \times AS_n$.

A *potential product* corresponds to any tuple $(s_1,...,s_n) \in D$.

In the previous example, the set of potential products D is comprised of 32 different potential products. $S_1 = (sel, sel, sel, sel, 2, 256)$ and $S_2 = (sel, rem, sel, rem, 1, 128)$ are examples of potential products. However, not all of them are *SPL products*, i.e. products that effectively can be built in the SPL. To define the set of all the SPL products, a first option that consists of enumerating all of them is immediately discarded when the number of SPL products shoots up. A set in general can be defined by a list of its elements or by a subset that satisfies a condition or constraint. SFMs use constraints to define the set of SPL products as a subset of the set D of potential products. An SFM has a set of *relationships* $R = \{R_1,...,R_i\}$ that sets the conditions a potential product must necessarily fulfil to be an SPL product. So the set of SPL products is defined as follows:

Definition 5.2 - Set of SPL products.
Let $R = \{R_1,...,R_i\}$ be the set of relationships in an SFM and D its set of potential products. The set of SPL products defined by the SFM is:

$$P = \{(s_1,...,s_n) \in D | R_1 \wedge ... \wedge R_i\}$$

It might be the case that there exist no product satisfying all the relationships and therefore $P = \emptyset$. In this case, it is said that the SFM is *void* or *invalid*.

The kinds of constraint that we propose for SFMs are the same than those used for FMs. Table §5.2 shows a list of constraints that can be used to represent the relationships in an SFM. So for example, if A is a parent feature linked by a set relationship with three child features A, B and C and affected by cardinality C_1, the following constraint and set of SPL products can be defined:

$$P = \{(s_R, s_A, s_B, s_C, s_{C_1}, s_{Mem}) | set_3(s_R, s_{C_1}, s_A, s_B, s_C)\}$$

Kinds of relationship constraints	
$mandatory(s_p, s_c)$	$\equiv s_p = s_c$
$optional(s_p, s_c)$	$\equiv s_c = sel \Rightarrow s_p = sel \wedge s_p = rem \Rightarrow s_c = rem$
$set_2(s_p, s_c, s_{c_1}, s_{c_2})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_2(s_c, s_{c_1}, s_{c_2}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem$
$set_3(s_p, s_c, s_{c_1}, s_{c_2}, s_{c_3})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_3(s_c, s_{c_1}, s_{c_2}, s_{c_3}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem \wedge s_{c_3} = rem$
...	...
$depends(s_1, s_2)$	$\equiv s_1 = sel \Rightarrow s_2 = sel$
$excludes(s_1, s_2)$	$\equiv \neg(s_1 = sel \wedge s_2 = sel)$

$^\dagger numSelChild_i(s_c, s_{c_1}, ..., s_{c_n})$ is a predicate that is true whenever the number of selected states in $s_{c_1}, ..., s_{c_n}$ coincides the cardinal s_c.

Table 5.2: Kinds of relationship constraints in a SFM

An SFM also stores a configuration which comprises all the decisions made by one or more users in a given moment. Decisions are collected in any order, supporting a parallel configuration process. These user decisions set a partition on the set of potential products D in a SPL: those that satisfy user decisions and those that do not. Following this criterion, we define a configuration as follows:

Definition 5.3 - Configuration.
Let $U = \{U_1, ..., U_j\}$ be a set of constraints defined on D, describing the user decisions in an SFM. A configuration C_U is defined as the set of potential products that satisfy all the user decisions:

$$C_U = \{(s_1, ..., s_n) \in D | U_1 \wedge ... \wedge U_j\}$$

Users can make two kinds of decisions as shown in Table §5.3. A user can either make a *choose decision*, that assigns a state for one element, or a *discard decision* that avoids an element having a certain state. So for example, $U_1 = choose(s_B, sel)$ identifies a user selecting feature B; $U_2 = discard(s_{Mem}, 128)$ identifies a user refusing a 128Kb bandwidth. With these two kinds of decisions, users delimit the set of potential products by means of successive refinements.

In case that user decisions contradict each other, as two users choosing selected and removed states for the same feature for example, then the set $C_U = \emptyset$. This situation is known as a *contradictory configuration*.

Kinds of decision constraints	
$choose(s_e, S)$	$\equiv s_e = S$
$discard(s_e, S)$	$\equiv s_e <> S$

Table 5.3: Kinds of decision constraints in a SFM

From the configuration and the set of SPL products, it can be determined the set of *compatible products*, i.e. the subset of SPL products that satisfy the criteria established by user decisions. The set of compatible products is defined as follows:

Definition 5.4 - Set of compatible products.
Let P be the set of SPL products defined by the relationship constraints $R_1, ..., R_i$, and let C_U be a configuration defined by the user decision constraints $U_1, ..., U_j$. The set of compatible products (CP) is defined as:

$$CP = P \cap C_U = \{(s_1, ..., s_n) \in D | R_1 \wedge ... \wedge R_i \wedge U_1 \wedge ... \wedge U_j\}$$

With the above definitions, we are able to represent an SFM in a compact manner as follows:

Definition 5.5 - Stateful Feature Model.
An SFM can be represented by a 4- tuple (E, D, R, U) such that E is the set of elements in the SFM, D is the set of potential products, R is the set of relationships and U is the set of user decisions.

From the information contained in this tuple, the set of potential, SPL and compatible products can be deduced. Next we introduce three concepts that can be defined on top of SFMs: SFM states, element states and automatic decisions.

5.3.1 SFM states

From the set of compatible products (CP), it is possible to identify singular situations or *SFM states*:

- Initial state: an SFM is in its initial state when there exist no user decisions, i.e. $U = \emptyset$. In this case, the configuration coincides with the space of potential states ($C_U = D$) and therefore the set of compatible products coincides the set of products, i.e. $CP = P$.
- Final state: an SFM is in a final state when it only determines one compatible product, i.e. $|CP| = 1$. When a final state is reached, it can be affirmed that a product satisfying all the user decisions has been found. The final adjective comes from the inability of the CP set to evolve in a manner that the set of compatible products is reduced even more.
- Intermediate state: A state is intermediate if $U \neq \emptyset$ and $|CP| > 1$. This is the most common state in an SFM and it corresponds to a situation in which users may still made decisions.
- Invalid state: if $CP = \emptyset$ at any moment, an SFM is said to reach an invalid state. This state can be reached because the set of SFM products is empty ($P = \emptyset$), or the configuration is contradictory ($C_U = \emptyset$), or there exist no compatible product satisfying the user decisions ($P \cap C_U = \emptyset$).

An SFM state makes reference to the instant in which the configuration process is. This way, the configuration process starts with an SFM in its initial state. User decisions provoke the SFM state to change to an intermediate state. A final state can be reached if there exist only one product satisfying all the user decisions. In case a configuration defines no compatible product, an invalid state is reached.

5.3.2 Element states and automatic decisions

From an SFM, it is possible to extract relevant information about the elements and states on which users might still make decisions without reaching an invalid state. For this purpose, we define the concept of *element state* as the set of all the states an element has for every compatible product in CP.

Definition 5.6 - Element State.
Let E_i be an element in an SFM and CP the set of compatible products. The element state for that element E_i is defined as follows:

$$State(E_i) = \bigcup_{s \in CP} \pi_i(s)$$

Being $\pi_i(s)$ the projection function that extracts the ith element from a tuple s that represents a compatible product.

So for example, let us consider the following set of compatible products:

$$E = \{F_R, F_A, F_B, F_C, C_1, Mem\}$$
$$CP = \{P_1, P_2, P_3\}$$
$$P_1 = \{sel, rem, sel, sel, 2, 256\}$$
$$P_2 = \{sel, rem, rem, sel, 1, 128\}$$
$$P_3 = \{sel, rem, sel, sel, 2, 128\}$$

The following element states are obtained from these compatible products:

$$State(F_R) = \{sel\}, \quad State(F_A) = \{rem\}, \quad State(F_B) = \{sel, rem\},$$
$$State(F_C) = \{sel\}, \quad State(C_1) = \{1, 2, \}, \quad State(Mem) = \{128, 256\}$$

An element state is used to infer what we call the set of *automatic decisions* (A). An automatic decision is the one that if it is made by a user, the set of compatible products remains unaltered. This way, if $A = \{A_1, ..., A_k\}$ represents the set of automatic decisions, and being C_A the subset of all the potential states that satisfy such automatic decisions:

$$C_A = \{(s_1, ..., s_n) \in D | A_1 \wedge ... \wedge A_i\}$$

it is possible to affirm that

$$CP \cap C_A = CP \Rightarrow CP \subseteq C_A$$

This set of automatic decisions is used to assist users in the decision making process, avoiding as far as possible to reach for invalid states. Automatic decisions can be calculated from the element states. Let E_i be an element in an SFM having AS_i as available states and $State(E_i)$ as its corresponding element state. Depending on the element state, an automatic decision can or cannot be inferred as follows:

- $State(E_i) = AS_i$: in this case, the element is said to be in an *undecided* state. It means that no user has made any decision that affects it and no automatic decision can be made on it.
- $|State(E_i)| = 1$: in this case, it can be interpreted that an automatic decision has been made on E_i to choose the unique state in $State(E_i)$
- $|AS_i| > |State(E_i)| > 1$: in this case, it can be interpreted that all the states that are not in $State(E_i)$ have been discarded. It means that those states in the set $AS_i - State(E_i)$ can be automatically discarded.
- $State(E_i) = \emptyset$: this situation can only be given if $CP = \emptyset$ in which case it does not make sense to calculate any automatic decision since there is no compatible product.

Each automatic decisions obtained following the above criteria must be checked if it already exists as a user decision, in which case they are disposed. For the previous example, the following set of automatic decisions can be inferred:

$$A = \{choose(s_A, sel), choose(s_C, sel), discard(s_{C_1}, 3)\}$$

From here and on, attributes are left out of the next steps in this dissertation. Our next contributions are given for features and cardinalities as the only elements in an SFM. Attributes can have discrete and continuous domains. Discrete domains can be treated like cardinalities, but continuous domains might introduce problems that have not been explored in this dissertation. However our proposal is built thinking in their addition as a future work as approached in Section §A.3.

5.4 STATEFUL FEATURE DIAGRAMS

Stateful feature diagrams are a graphical notation of SFMs based on feature diagrams [76]. The representation of elements and relationship constraints is the same that FMs: a tree-like structure where features are boxes linked by different kinds of lines that represent the relationships among features. Cardinalities are also drawn together with the corresponding set relationship.

User and automatic decisions are not explicitly depicted in the diagram. Instead of it, element states are drawn. If a feature state is $\{sel\}$, it is drawn as a tick (✓) within a

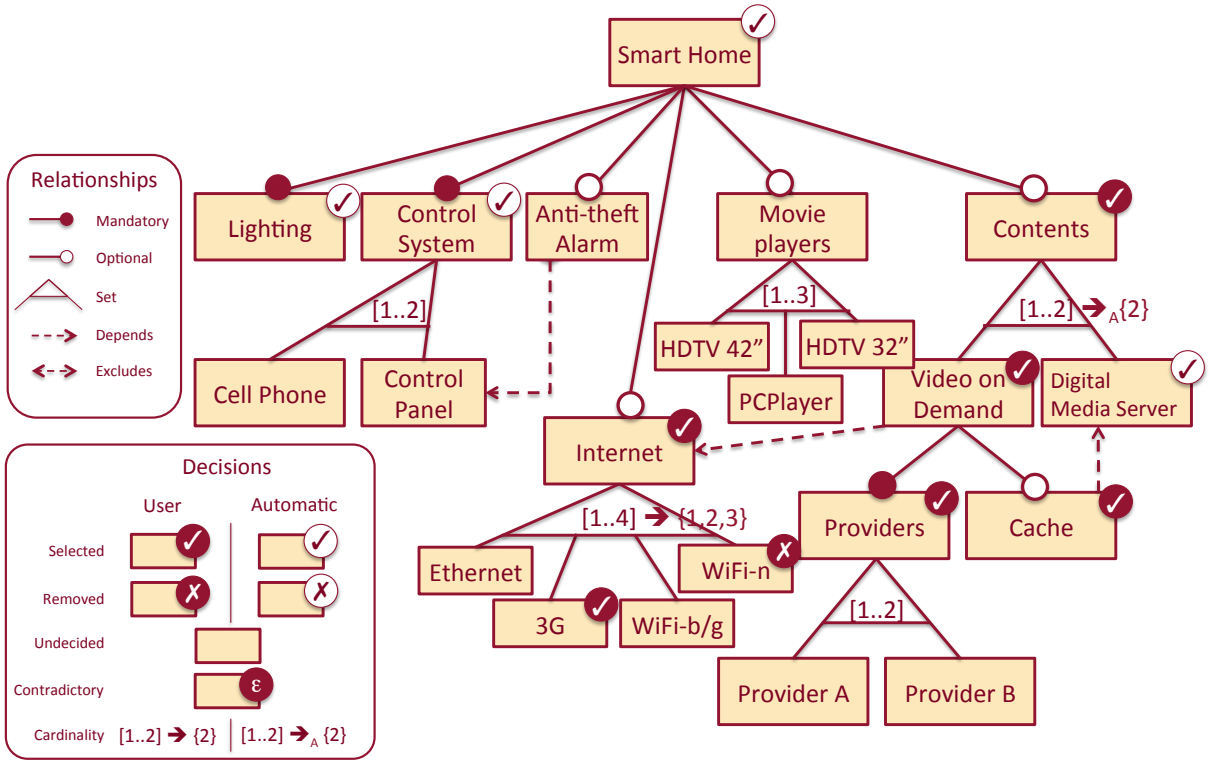


Figure 5.3: An example of a Stateful Feature Diagram

circle in a corner of the feature box; if a feature state is $\{rem\}$, a cross (✗) is used instead; if a feature is undecided, no specific mark is assigned; if two or more users have made contradictory decisions on a feature, it is marked as ε. It the feature state is inferred by user decisions, then a filled circle is used; if it can only be inferred by automatic decisions, a non-filled circle is used instead.

Decisions on cardinalities are drawn as $C_d \rightarrow C_s$ where C_d is the cardinality domain and C_s is the element state for that cardinality. For example a cardinality $[1..3]$ whose state has been set to $\{2\}$ is drawn as $[1..3] \rightarrow \{2\}$. If state 2 is removed but no other state has been chosen, it is represented as $[1..3] \rightarrow \{1,3\}$. Just in case the cardinality state is inferred only from automatic decisions, $C_d \rightarrow_A C_s$ is used to remark that its state has changed due to automatic decisions. Figure §5.3 shows an example of a stateful feature diagram.

Since attributes have been left out of the scope of this dissertation, our proposal on stateful feature diagrams does not incorporate them.

5.5 STATEFUL FEATURE METAMODEL

A metamodel is a rigorous definition of a model that describes their main concepts, relationships and rules. We propose the *Stateful Feature Metamodel (SFMM)* as a rigorous definition of SFMs. Besides rigour, metamodels can be used as development artefacts in *Model-driven Engineering (MDE)*. It enables to build analysis tools by the definition of transformations into other declarative languages which can be used to reason about them.

Obtaining a metamodel is a design exercise. So we firstly propose a list of objectives the design must satisfy according to the abstract model presented in Section §5.3:

Objective 1. Representing a set of elements, each of which has a set of available states.

Objective 2. Representing the set of SPL products by means of a set of relationship constraints.

Objective 3. Representing decisions as a set of constraints, distinguishing among user and automatic decisions.

Objective 4. Enabling the computation of element states according to user and automatic decisions.

Objective 5. Proposing an extensible metamodel that supports the future addition of new kinds of relationships and elements.

Figure §5.4 depicts our proposal of an SFMM using a UML class diagram that satisfies the above objectives. The design decisions that have led to the SFMM are described in detail in Annex §A.

An SFM is represented by a `StatefulFeatureModel` instance. It is a container of `Element`, `Relationship` and `Configuration` instances. The elements in an SFM can be either `GenericFeature` or `Cardinality` instances. There exists a class for each kind of relationship such as mandatory, optional, set, requires and excludes relationships. A `GenericFeature` class is defined to distinguish between the root and remaining features. It is used to avoid the incorrect use of the root feature in relationships such as the root feature being the child of a relationship or an exclusion between a root and any other feature that provokes a dead feature.

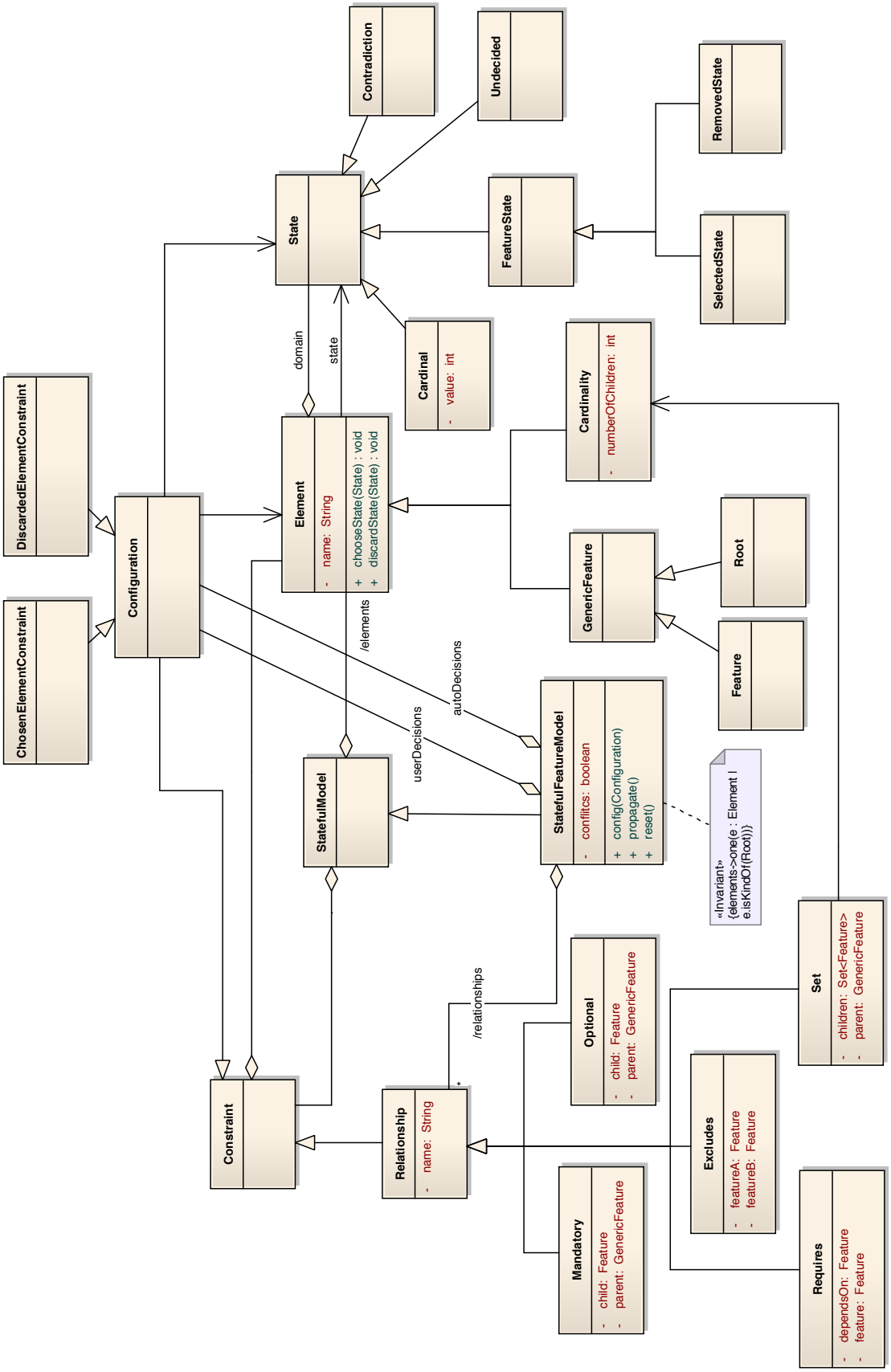


Figure 5.4: Stateful Feature Metamodel in UML format

The decisions can be defined in terms of an element selection (`ChosenElementConstraint` class) or discarding (`DiscardedElementConstraint` class). An SFM distinguishes between user and automatic decisions, storing them in two different aggregates. Decisions are introduced in the model by means of the model edition operations defined in Section §5.5.1. Remaining operations, such as those to build an SFM from scratch or to modify elements and relationships have been left out of the Figure since they are not relevant for the purpose of this dissertation.

Every element has a domain and a current state. The domain is the set of available states. Each kind of element has its own element-specific available states: `SelectedState` and `RemovedState` for features and `Cardinal` for cardinalities. An element state is immediately updated when a user decision is introduced into an SFM. Initially, every element state contains an `Undecided` state instance to indicate that no decision has been made on them. As decisions are added, the state of the affected elements changes to represent the decisions. In case a contradiction is found among decisions, such as a feature that is selected and removed at the same time, the `Contradiction` state instance is set as the state for for the affected elements.

Last, there exist some generalisation points to increase the extensibility of the metamodel. A `StatefulModel` class is defined in case the constraint-element-state structure wants to be reused for other models; relationships and decisions are considered as two different kinds of `Constraint`. In case we want to add a new kind of constraint that is not covered by relationships and decisions, it can be done as an extension of this class.

As a further reading, Annex §A details the design decisions that have led us to propose this metamodel.

For the creation and manipulation of SFMs, a set of model edition operations must be defined. From all the operations that could be defined for this purpose, in next Section we just show those operations that modify the configuration of the SFM, which are relevant for analysis purposes.

5.5.1 Model edition operations

Model edition operations are defined to manipulate user and automatic decisions, changing the SFM state and therefore providing the dynamic behaviour in SFMs. Following we present three model edition operations:

Operation 1 - Setting user decisions.
This operation adds a set of user decisions to an SFM. It receives an SFM and a set

of decision constraints as inputs and produces a new SFM where the user decisions are added. The set of automatic decisions is not recalculated as a consequence of this operation. An example of this operation is shown in Figure §5.5. Since the output is a new SFM, it can be chained to itself or to other analysis operations.

This operation is represented by a method whose prototype is `SFM setUserDecisions(SFM inputSFM, Set<Decision> configSet)`.

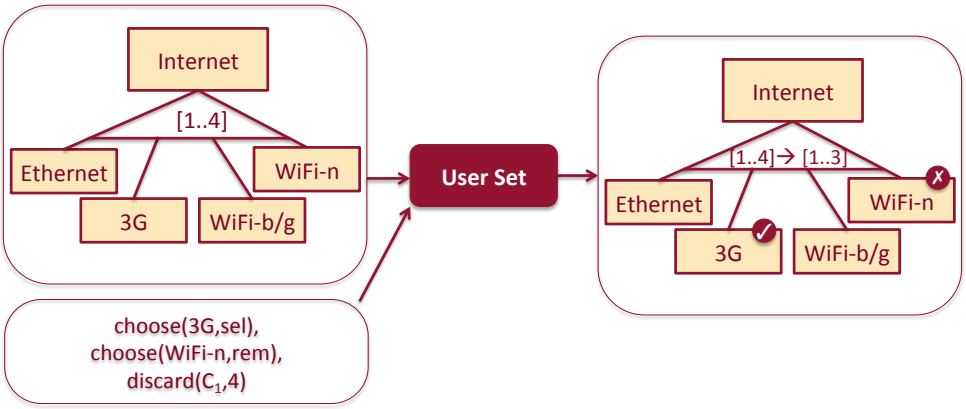


Figure 5.5: An example of a user configuration

Operation 2 - Setting automatic decisions.
This operation adds a set of automatic decisions to an SFM. It receives an SFM and a set of user decisions as inputs and produces a new SFM where the automatic decisions are added. This method is designed to be used by an algorithm that computes the set of automatic decisions and by automated analysis tools that need to simulate decisions for analysis purposes. It must not be used by users. The prototype for this operation is a method `SFM setAutoDecisions(SFM inputSFM, Set<Configuration> configSet)`. An example of this operation is shown in Figure §5.6.

Operation 3 - State reset.
This operation removes all the user and automatic decisions from the SFM to restore the initial state of the model. As a consequence, the state is updated and all the elements are assigned an undecided state. The prototype for this operation is a method `SFM reset(SFM inputSFM)`. Figure §5.7 shows an example of this operation.

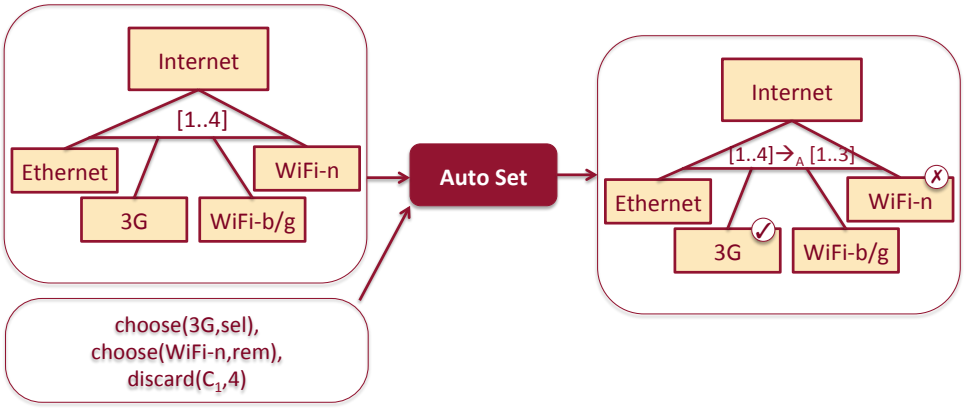


Figure 5.6: An example of a user configuration

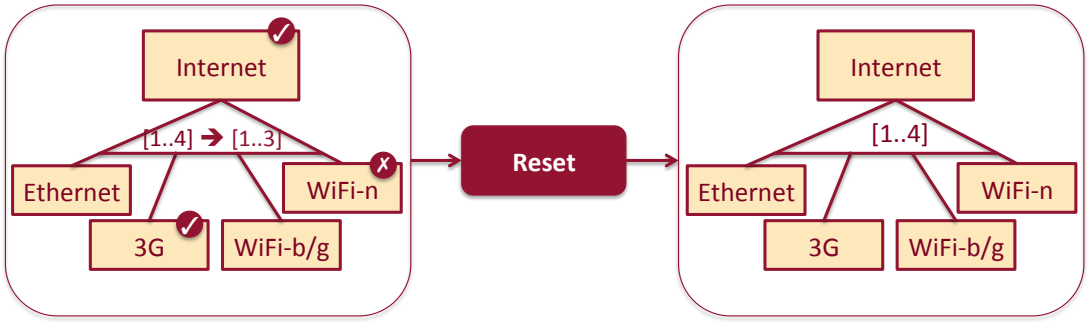


Figure 5.7: An example of state reset operation

5.6 SUMMARY

In this Chapter we have presented SFMs as a new kind of model that enables fully-configurable FMs. SFMs incorporate the concept of element states to describe the dynamic behaviour of SFMs which is a novel approach in the description of products in a SPL. SFMs distinguish among potential products as any combination of element states, SPL products as those potential products that satisfy the relationship contraints, and compatible products as the SPL products that satisfy the configuration constraints. They also allow to work with multiple users making decisions on an SFM at the same time and distinguishes between user and automatic decisions.

Stateful feature diagrams are proposed to graphically describe SFMs. They are inspired in feature diagrams where states are added to represent user and automatic decisions.

Besides the rigorous definitions given in this Chapter for SFMs, we define them

in terms of an UML metamodel. Metamodels enable their use in MDE tools such as the prototype that we build to demonstrate the feasibility of our proposal in Chapter §10. The SFMM takes extensibility as a major concern. The extension mechanisms allow the future exploration of the impact of attributes which have been left out of this proposal. We propose a set of model edition operations that enables the manipulation of decisions. They will play a key role in the formal specification framework that we propose in this dissertation.

In next Chapters we set the basis to automate the analysis of SFMs, relying on logics to obtain important information from these models. The metamodel presented in this paper will play a key role in the implementation of the analysis techniques.

6

Automated analysis of Stateful Feature Models

I will be so brief I have already finished

Salvador Dali (1904–1989),
Painter, artist

The definition of a new kind of model such as SFMs suggests the proposal of techniques for their automated analysis. This Chapter defines the scope of the Automated Analysis of Stateful Feature Models (AASFM) and provides a general schema for it in Section §6.1. In Section §6.2 we propose a taxonomy of analysis operations for the AASFM. Section §6.3, proposes a generic logical representation for SFMs that is used as an intermediate model to ease the mapping to DAPs in the following Chapters for the formalisation of the AASFM. Last, Section §6.4 summarises the contributions in this Chapter.

6.1 General Schema

The proposal of a new model suggests the adaptation of the AAFM to the new characteristics of SFMs. This is why we propose what we call the AASFM which we, inspired by Benavides et al. [14], define as follows:

Definition 6.1 - Automated Analysis of Stateful Feature Models (AASFM).
The AASFM can be defined as the computer-aided extraction of information from SFMs by automated means.

The main innovation in SFMs resides in their fully-configurable capabilities. Apart from this, SFMs retain all the expressiveness of FMs and CMs. This implies that the set of analysis operations for the AASFM must contain at least all the operations defined to date for the AAFM.

Such as for the AAFM, we propose the transformation of SFMs into a logical language that encodes the information in the form of logical formulas. A logical language confers an operational semantics to SFMs while enabling logical reasoning which allows the derivation of conclusions from the logical formulas. The mechanisms for logical reasoning are precisely defined and allow the automation of reasoning and therefore the automation of the AASFM. Besides analysis capabilities, these transformations provide a translational semantics to SFMs.

There is a wide catalogue of logical languages that can be used for the AASFM: first-order logics, CSPs, binary decision diagrams, temporal logics, description logics, etc. Independently of the particular language used to reason, there are three main kinds of reasoning: deduction, abduction and induction. All the operations proposed for the AAFM can be interpreted as cases of deduction and abduction [85]. Abduction is used to obtain explanations why certain situations are given such as dead features, false-optional features or a void FM. Deduction is used to extract any other information from a FM such as counting the number of products, obtaining a list of products, etc.

Instead of proposing a formalisation of the AASFM based on a specific logic, we propose a formalisation as independent of logics as possible. For that purpose, we interpret SFMs as DAPs, that can be used to perform deductive or abductive-based analysis operations. Thanks to this approach, we are able to identify which AASFM operations correspond to basic DAP operations, defining a set of *basic operations* for the AASFM. This way, remaining AAFM operations can be interpreted in the AASFM as a

combination of such basic operations in what we call *compound operations*. This division allows us to define a taxonomy of analysis operations that we propose in Section §6.2.

When transforming SFMs into DPs and APs for their analysis, we realise that there exist many common elements between both transformations. For this reason we propose a *Stateful Feature Model Problem (SFMP)* as a generalisation of DAPs. An SFMP acts as an indirection level between SFMs and DAPs that collects the common artefacts in DAPs and the common mapping rules in the transformation from SFMs. It can be seen as an mere engineering instrument to avoid the repetition of mapping rules.

All these contributions result in a AASFM general schema that is depicted in Figure §6.1.

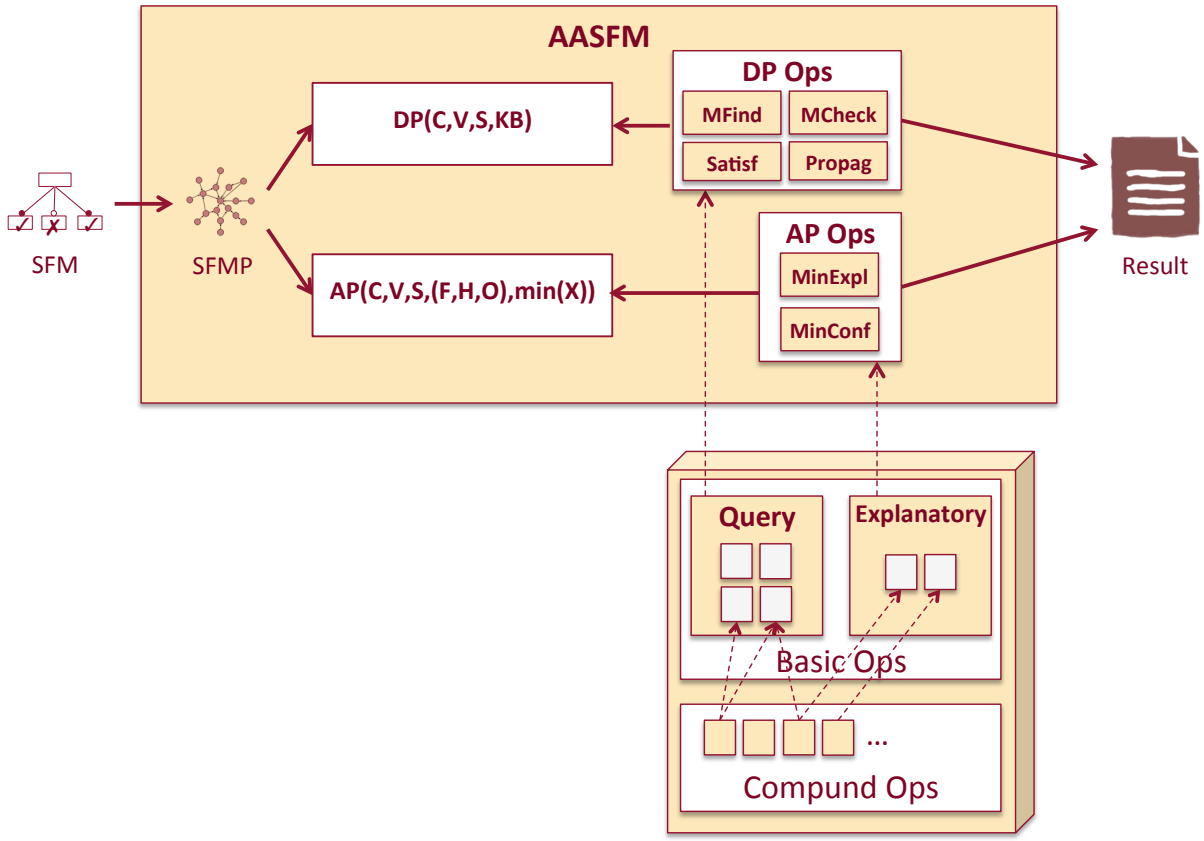


Figure 6.1: General schema of the AASFM

6.2 A TAXONOMY OF OPERATIONS

The approach that we follow to formalise the AASFM allows us to propose the following taxonomy of operations:

- Basic operations: if a SFM is interpreted as a DP and an AP, a basic operation is solved using one and only one reasoning operation (such as solutions, satisfiability, minimal explanations, etc.). Depending on the kind of reasoning operation that is used, two subtypes of basic operations are distinguished:
  - Query operations: the goal of a query operation is to extract implicit information from a SFM and make it explicit. They are implemented as deductive operations on a DP.
  - Explanatory operations: The goal of explanatory operations is obtaining explanations why the user decisions or the relationships provoke a certain behaviour. They are implemented as abductive operations on an AP.
- Compound operations: a compound operation combines basic operations to perform higher-level operations.

The main benefit of this taxonomy is that only basic operations need to be given a formal semantics, while remaining operations are defined as a combination of them. The composability of model edition operations together with the combination of basic operations allow to propose a wide catalogue of analysis operations. In this dissertation we propose a catalogue of AASFM operations that corresponds with most of the AAFM operations defined up to date. We also propose new analysis operations that were not considered for the AAFM which are now possible using SFMs.

6.3 A LOGICAL REPRESENTATION OF STATEFUL FEATURE MODELS

Representing SFMs in terms of logics implies the definition of all the needed elements for logical reasoning. Many different logics can be used to encode and to reason on a model. There exist commonalities among logics. In particular, any logic requires a language with a formal syntax and a precise semantics, a notion of logical entailment

and rules for manipulating expressions. We propose a logical framework that we coin as SFMP, which can be defined as follows:

Definition 6.2 - Stateful Feature Model Problem (SFMP).
A SFMP is a 7-tuple $SFMP(C,V,D,R,U,A,S)$ such that C is a set of constants representing elements and the element-specific states; V is a set of variables representing the state for each element; D is a set of formulas that define the element available states or *domains*; R, U and A are sets of formulas for relationships, user and automatic decisions respectively; and S is a semantics for the formulas used in R, U and A.

An SFMP relies on FOL. We use natural numbers, algebraic functions and any other element that are not defined in classical FOL wherever they ease the representation. The $\models$ symbol is used along this dissertation to represent an inference operation applying the corresponding semantics to the logical language. Despite the use of FOL, we think that our approach can be easily adapted to be represented in different kinds of logics that can benefit from the use of automated reasoning tools. It will be the work of specific implementations to find the most suitable logic for this representation.

The inference rules to obtain semantic and syntactic consequences are undefined in this representation. Depending on the kind of analysis operation to perform, the reasoning problem to solve may change and so the reasoning mechanisms. In Chapters §7 and §8 we interpret the different SFM analysis operations as DPs and APs, using the SFMP representation as a starting point.

Next we propose a mapping to create a SFMP from a SFM. We guide the mapping through an example in Figure §6.2.

6.3.1 Constants (C)

A constant in C represents a static item in a SFM. In this case they represent the elements (E) and states (S) in a SFM. Caps letters are used for each constant. The following set of constant results for the given example:

$$C = \{F_{root}, F_A, F_B, F_C, F_D, F_E, C_1, sel, rem, 1, 2, 3\}$$

6.3.2 Variables (V)

A variable in V represents a dynamic item in a SFM. Variables are used to represent the current element states. So every element in $E_i \in E$ has its corresponding variable

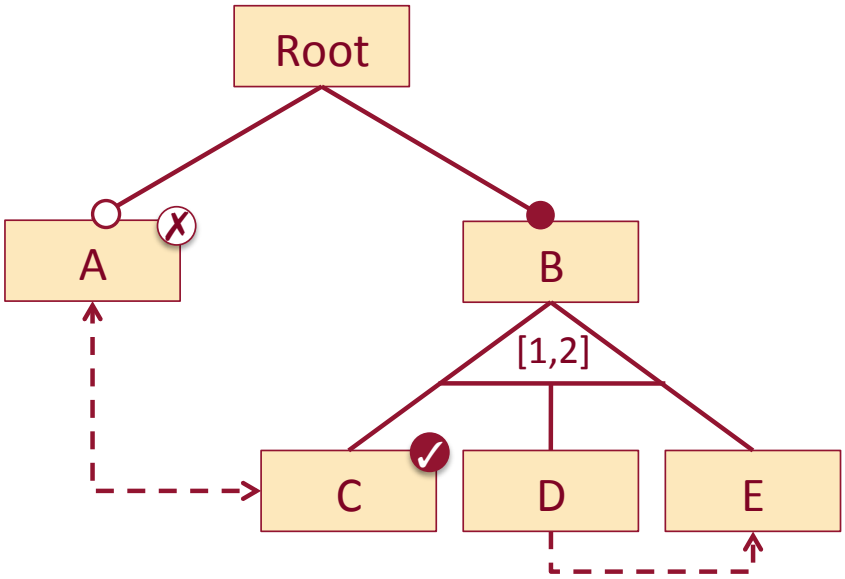


Figure 6.2: A stateful feature diagram example

$s_{E_i} \in V$. The following set of variables results for the given example:

$$V = \{s_{F_{root}}, s_{F_A}, s_{F_B}, s_{F_C}, s_{F_D}, s_{F_E}, s_{C_1}\}$$

6.3.3 Domains (D)

The set of domains defines the available states in terms of logical predicates. For this purpose, any $s_j \in AS_{e_i}$ such that AS_{E_i} is the set of available states for an element $E_i \in E$, is represented by a $state(E_i, S_j)$ predicate such that E_i and S_j are the corresponding element and state constants.

In a product, an element can only have one state from the available ones. This approach is followed to represent the domain as a mutual exclusion of the available states[1]. So for example, the available states for a feature are described in the domain as a logical formula in the following form:

$$state(F_i, sel) \oplus state(F_i, rem)$$

Initially, neither $state(F_i, sel)$ nor $state(F_i, rem)$ can be confirmed to be true or false, so the above formula represents the undecided state itself. If $state(F_i, sel)$ and $state(F_i, rem)$

[1]The $\oplus$ operator corresponds to mutual exclusion and can be defined as follows: $a \oplus b \equiv (p \wedge \neg q) \vee (\neg p \wedge q)$.

are true at the same time, then a contradiction state arises. If any of the states is discarded, then the other state can be assumed to be true. For example if a feature cannot be selected then it can be immediately concluded that the feature must be removed:

$$state(F_i,sel) \oplus state(F_i,rem) \wedge \neg state(F_i,sel) \models state(F_i,rem)$$

A root feature can only have the selected state so that its domain is defined simply as $state(F_{Root},sel)$.

Cardinalities can be represented by a mutual exclusion of the available cardinals as for features. However, and due to the representation that we propose later for set relationships, we model them in a different manner. The widest possible range of cardinals in cardinalities is $[0..n]$ such that n is the number of child features in the set relationship. If a more restrictive range is defined, the corresponding domain must not only define the valid cardinals but those cardinals that cannot be given. So for example, the domain formula for a set relationship with three child features and a cardinality $\{1,2,4\}$ results as follows:

$$(state(C,1) \oplus state(C,2) \oplus state(C,4)) \wedge \neg state(C,0) \wedge \neg state(C,3)$$

Lastly, the domain also includes a formula to link element constants with the corresponding state variables as follows:

$$state(E_1,s_{E_1}) \wedge state(E_2,s_{E_2}) \wedge ... \wedge state(E_n,s_{E_n})$$

Following the above rules, we obtain this set of domain formulas results from the example:

$$\begin{aligned} D = \{ & state(F_{root},sel), \\ & state(F_A,sel) \oplus state(F_A,rem), \\ & state(F_B,sel) \oplus state(F_B,rem), \\ & state(F_C,sel) \oplus state(F_C,rem), \\ & state(F_D,sel) \oplus state(F_D,rem), \\ & state(F_E,sel) \oplus state(F_E,rem), \\ & (state(C_1,1) \oplus state(C_1,2)) \wedge \neg state(C_1,3), \\ & state(F_{Root},s_{F_{Root}}) \wedge state(F_A,s_{F_A}) \wedge state(F_B,s_{F_B}) \wedge state(F_C,s_{F_C}) \wedge \\ & state(F_D,s_{F_D}) \wedge state(F_E,s_{F_E}) \wedge state(C_1,s_{C_1}) \} \end{aligned}$$

6.3.4 Relationships (R)

The set of relationships contains formulas to represent the relationships in a SFM. For each relationship in the SFM, one formula is added, depending on the kind of relationship:

- Mandatory: a parent feature `P` that has a mandatory relationship with a child feature `C` is represented by:

$$mandatory(F_P, F_C)$$

- Optional: a parent feature `P` that has an optional relationship with a child feature `C` is represented by:

$$optional(F_P, F_C)$$

- Depends: if a feature `A` depends on another feature `B`, the relationship is represented by:

$$depends(F_A, F_B)$$

- Excludes: two features `A` and `B` in mutual exclusion are represented by:

$$excludes(F_A, F_B)$$

- Set: let us consider a parent feature `P` in a set relationship with a set of child features `C1`,...,`Cj`. If the cardinality for the set-relationship is represented by a constant C_i, then the following formula is defined:

$$set_j(F_B, C_i, F_{C_1}, ..., F_{C_j})$$

The following set of relationships results for the given example:

$$\begin{aligned} R = \{&optional(F_{root}, F_A),\\ &depends(F_D, F_E),\\ &mandatory(F_{root}, F_B),\\ &excludes(F_A, F_C),\\ &set_3(F_B, C_1, F_C, F_D, F_E)\} \end{aligned}$$

6.3.5 User and automatic decisions (U, A)

These two sets of decisions contain formulas that represent the user and automatic decisions. Each user and automatic decision is mapped into a logical formula in the following form:

- Choose decision: a decision that chooses a state S_j for an element E_i is represented by:
$$choose(E_i, S_j)$$
- Discard decision: a decision that discards a state S_j for an element E_i is represented by:
$$discard(E_i, S_j)$$

The following set of relationships results for the given example:

$$U = \{choose(F_C, sel)\}$$
$$A = \{choose(F_A, rem)\}$$

6.3.6 Semantics (S)

Logical formulas have no meaning unless a semantics is given to its symbols. In this case, every formula used to represent relationships and decisions are given a meaning describing how they affect the SFM state. The way each kind of relationship affects the state is defined in terms of *state* predicates as shown in Table §6.1. The terms used in the formulas are variables such as f_c or f_p. They are substituted in each relationship by the corresponding constants. So for example, the semantics for $mandatory(F_{root}, F_B)$ is $state(F_{root}, sel) \Leftrightarrow state(F_B, sel) \wedge state(F_{Root}, rem) \Leftrightarrow state(F_B, rem)$.

As it can be noticed, the semantics is the same for every SFM, which makes it an invariant artefact in a SFMP.

6.3.7 Traceability table

A traceability table contains a bijective relationship between SFM artefacts and constants and variables in a SFMP. It keeps the traceability between SFMPs and SFMs, allowing to interpret any result of logical reasoning in terms of SFM artefacts. For example during the generation of D, R, U and A this table is used to find the adequate

Semantics	
Syntax	**Semantics**
$mandatory(f_p, f_c)$	$state(f_p, sel) \Leftrightarrow state(f_c, sel) \wedge$
	$state(f_p, rem) \Leftrightarrow state(f_c, rem)$
$optional(f_p, f_c)$	$state(f_c, sel) \Rightarrow state(f_p, sel) \wedge$
	$state(f_c, rem) \Rightarrow state(f_p, rem)$
$set_n(f_p, c_i, f_{c_1}, ..., f_{c_n})^{\dagger}$	$state(f_p, sel) \Leftrightarrow numSelChild_n(c_i, f_{c_1}, ..., f_{c_n}) \wedge$
	$(state(f_p, rem) \Rightarrow \forall k(state(f_{c_k}, rem))$
$depends(f_a, f_b)$	$state(f_a, sel) \Rightarrow state(f_b, sel)$
$excludes(f_a, f_b)$	$state(f_a, sel) \Leftrightarrow \neg state(f_b, sel) \wedge$
	$state(f_b, sel) \Leftrightarrow \neg state(f_a, sel)$
$choose(e, s)$	$state(e, s)$
$discard(e, s)$	$\neg state(e, s)$

† The $numSelChild_j$ predicate is defined for each number j of child features. For example:

$$\begin{aligned} numSelChild_2(c_i, f_{c_1}, f_{c_2}) \equiv \quad & state(c_i, 0) \Leftrightarrow (state(f_{c_1}, rem) \wedge state(f_{c_2}, rem)) \\ & state(c_i, 1) \Leftrightarrow (state(f_{c_1}, sel) \oplus state(f_{c_2}, sel)) \wedge \\ & state(c_i, 2) \Leftrightarrow (state(f_{c_1}, sel) \wedge state(f_{c_2}, sel)) \end{aligned}$$

Table 6.1: Semantics for a SFMP

constants and variables that refer to elements and states in the SFM. Table §6.2 presents the traceability table for the running example.

6.4 SUMMARY

In this Chapter we present the AASFM as a set of techniques that enables the automated extraction of information from SFMs. The AASFM must provide for a catalogue of analysis operations and a formalisation of every operation in the catalogue. We propose a taxonomy of operations that reduces the operations needed to formalise the AASFM to a subset of basic operations, while remaining operations are defined on top of them.

These basic operations can be interpreted as DAP operations, which claims for an interpretation of SFMs as DAPs. It is the purpose of Chapters §7 and §8 to present how to interpret a SFMP as a DP and APs respectively and to interpret basic operations

Traceability table			
Constants	**Elements and states**	**Variables**	**Element states**
F_{Root}	Root feature	$s_{F_{Root}}$	state of root feature
F_A	Feature A	s_{F_A}	state of feature A
F_B	Feature B	s_{F_B}	state of feature B
F_C	Feature C	s_{F_C}	state of feature C
F_D	Feature D	s_{F_D}	state of feature D
F_E	Feature E	s_{F_E}	state of feature E
C_1	Cardinality C1	s_{C_1}	state of cardinality C1
sel	selected state		
rem	removed state		
1	cardinal 1		
2	cardinal 2		
3	cardinal 3		

Table 6.2: Traceability table for constants and variables

as deductive and abductive operations. Once basic operations will be formalised, a catalogue of compound operations is proposed in Chapter §9.

In order to ease the transformation of SFMs into DAPs, we propose SFMPs as a generic logical representation of SFMs. We have proposed a representation that is as independent of a particular kind of logic as possible. This is an instrumental representation of SFMs that is used as the starting point to solve any analysis operations for the AASFM.

7

Interpreting query operations as deduction operations

Logic, like whiskey, loses its beneficial effect when taken in too large quantities.

Edward Plunkett (1878–1957),
Dramatist

In this Chapter we give a semantics for SFMs as DPs. This mapping enables the definition of basic query operations in terms of deduction operations. Firstly, Section §7.1 proposes a mapping from SFMPs to DPs. In Section §7.2 we propose three query operations that can be solved by means of three deductive operations. Lastly, Section §7.3 summarises the contributions of this Chapter.

7.1 SFMs AS DEDUCTION PROBLEMS

SFMs and FMs have commonalities for their automated analysis since they share many elements. The need for an automated extraction of implicit information remains for SFMs. Operations to detect errors such as dead features, counting or listing products or determining the number of products in which a feature appears make sense either for FMs or for SFMs.

The objective of deductive reasoning is obtaining conclusions from a certain knowledge represented in logical terms. The objective of query operation is making explicit an information that is implicitly modelled in a SFM. This definition of query operation fits in the goal of deductive reasoning. In this Chapter we identify which deductive operations make sense in the context of SFMs and we propose a set of basic query operations that can be solved using DP operations.

In order to apply deductive reasoning to SFMs, we need a representation of a SFM in terms of a DP. In Chapter §6 we proposed SFMPs as a logical representation of SFMs that eases the transformations to DPs and APs. In this case, SFMP are very close to the representation of DPs. They both define a syntax and a semantics. In both cases, constants are variables refer to real-world elements. However the logical sentences are structured in different ways. A SFMP divides the logical sentences in four groups of sentences domains, relationships, user and automatic decisions; meanwhile DPs store all the sentences in a unique KB. In a DP that represents a SFMP, all the sentences are stored in the KB indistinctly, while constants, variables and semantics remain unchanged. So a DP can be obtained from a SFMP applying the following mapping[1]:

$$SFMP(C,V,D,R,U,A,S) \mapsto DP(C,V,D \cup R \cup U \cup A,S)$$

Table §7.1 illustrates a DP for the stateful feature diagram example in Figure §6.2. The DP semantics, which is not represented in the example, remains unaltered from SFMPs (Table §6.1). The traceability table used for the SFMP also goes together with the DP. It is used to interpret the results from DP operations in terms of SFM artefacts. Next Section details the relationship between both kinds of problems.

[1] see Section §3.1 for a description of the DPs structure

Deduction problem
Constants (C)
$F_{root}, F_A, F_B, F_C, F_D, F_E, C_1$ $sel, rem, 1, 2, 3$
Variables (V)
$s_{F_{root}}, s_{F_A}, s_{F_B}, s_{F_C}, s_{F_D}, s_{F_E}$
knowledge Base (KB)
$state(F_{root}, sel)$, $state(F_A, sel) \oplus state(F_A, rem)$, $state(F_B, sel) \oplus state(F_B, rem)$, $state(F_C, sel) \oplus state(F_C, rem)$, $state(F_D, sel) \oplus state(F_D, rem)$, $state(F_E, sel) \oplus state(F_E, rem)$, $(state(C_1, 1) \oplus state(C_1, 2)) \wedge \neg state(C_1, 3)$, $state(F_{Root}, s_{F_{Root}}) \wedge state(F_A, s_{F_A}) \wedge state(F_B, s_{F_B}) \wedge$ $state(F_C, s_{F_C}) \wedge state(F_D, s_{F_D}) \wedge state(F_E, s_{F_E}) \wedge state(C_1, s_{C_1})$, $optional(F_{root}, F_A)$, $depends(F_D, F_E)$, $mandatory(F_{root}, F_B)$, $excludes(F_A, F_C)$, $set_3(F_B, C_1, F_C, F_D, F_E)$, $choose(F_C, sel)$, $choose(F_A, rem)$

Table 7.1: A DP representing the example in Figure §6.2

7.2 BASIC QUERY OPERATIONS

In order to propose a set of basic query operations that make sense from a deductive point of view, we enumerate the four deductive operations that are available in DPs. In order to interpret them as AASFM operations, we inspire in the catalogue of AAFM operations in [14] and adapt those operations to SFMs:

- Solutions: this operation searches for assignments of constants to variables that satisfy the logical formulas in the KB. In our case, an assignment binds states

to elements that satisfy all the domain, relationships and decision formulas. A valid assignment of states satisfies all the sentences in the KB: the domain, relationships and decisions. In terms of a SFM an assignment can be interpreted as a product that satisfies all the relationships, user and automatic decisions. Thus, this operation can be used for *products listing*.

- Satisfiability: this operation determines if there is at least one valid assignment of constants to variables. Since an assignment is identified with a product, satisfiability can be used to determine if there exists at least one product satisfying all the logical formulas. Depending on the set of user decisions, there are two possible scenarios: *i*) if it is non-empty, satisfiability helps to detect if there exists at least one product satisfying those decisions; *ii*) if the set of user decisions is empty, satisfiability detects if there exists at least one product in the SPL, i.e. the SFM is valid. We call *validation* to the AASFM operation that results from the application of the satisfiability operation.

- Inference: this operation determines if a logical sentence can be concluded from the DP. DPs rely on *state* predicates so conclusions obtained from the DP must be defined on their terms. For example, inference can be used to determine if $state(F_B, sel)$ can be concluded from the example in Table §7.1. It can be used to obtain automatic decisions based on existing user decisions, domains and relationships. The computation of automatic decisions on FMs is known as propagation, name that we keep for the AASFM.

- Solution checking: this operation checks if a given assignment is a solution for the DP. Since an assignment corresponds to a product, this operation can be used to check if a given product is valid or not. User configurations can be used instead of an input assignment to set the state for any element in the SFM. The resulting model can be checked if it is valid by means of satisfiability. Due to the existence of an alternative to perform model checking, and for the sake of keeping the set of basic operations minimal, we define no basic query operation in AASFM that uses model checking.

Based on this analysis we propose three basic query operations that can be solved using deductive reasoning: product listing, validation and propagation. Next we define a correspondence between these query operations and DP operations.

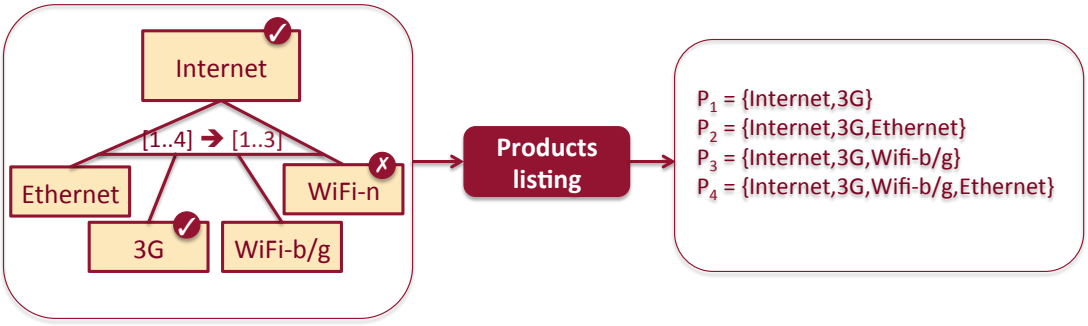


Figure 7.1: An example of products listing operation

7.2.1 Products listing

This operation lists all the products in a SFM that satisfy the current decisions and relationships. Figure §7.1 shows an example of this operation for an excerpt of the SHS FM. It corresponds to the solutions operation of a DP. A solution in a DP can be defined as a function $\mu : V \rightarrow C$. Each variable in a solution is assigned a constant that represents an available state for the element the variable represents. Only state constants such as *sel*, *rem* and cardinals are assigned to the variables, since the domain sets this restriction. Therefore, an assignment represents a product in the SFM. So the solutions operation searches for all the products that satisfy all the sentences in the KB: domains, relationships and decisions. From this rationale, the product listing operation can be interpreted in terms of the solutions operation as follows:

$$products(SFMP(C,V,D,R,U,A,S)) \equiv solutions(DP(C,V,D \cup R \cup U \cup A,S))$$

For the running example described in Figure §6.2, the following assignments are valid:

$$\mu_1 = \{s_{F_{Root}} \mapsto sel,\ s_{F_A} \mapsto sel,\ \ s_{F_B} \mapsto sel,\ s_{F_C} \mapsto sel,\ s_{F_D} \mapsto rem,\ s_{F_E} \mapsto sel,\ \ s_{C_1} \mapsto 2\}$$
$$\mu_2 = \{s_{F_{Root}} \mapsto sel,\ s_{F_A} \mapsto rem,\ s_{F_B} \mapsto sel,\ s_{F_C} \mapsto sel,\ s_{F_D} \mapsto rem,\ s_{F_E} \mapsto rem,\ s_{C_1} \mapsto 1\}$$
$$\mu_3 = \{s_{F_{Root}} \mapsto sel,\ s_{F_A} \mapsto rem,\ s_{F_B} \mapsto sel,\ s_{F_C} \mapsto sel,\ s_{F_D} \mapsto rem,\ s_{F_E} \mapsto sel,\ \ s_{C_1} \mapsto 2\}$$

If the DP contains any contradictory logical sentence then the result is an empty set of products. The results from this operation is obtained in terms of constants and variables. In order to interpret the results in terms of SFM elements and states, the traceability table is used. Each feature variable s_{F_i} corresponds to a feature constant F_i, which is conveniently linked to a feature in the original SFM in the traceability table. Since products are usually represented only by the selected features, the corresponding

products for the above assignments are:

$$P_1 = \{Root, B, E\}$$
$$P_2 = \{Root, B, D, E\}$$
$$P_3 = \{Root, B, C, E\}$$

7.2.2 Validation

A validation operation checks if a SFM is valid or not. It means to check if the decisions within the SFM satisfies all the relationships. In terms of a DP where the set of domains, relationships and decisions form the KB, a validation operation corresponds to the satisfiability operation for the DP:

$$valid(SFMP(C,V,D,R,U,A,S)) \equiv isSatisfiable(DP(C,V,D \cup R \cup U \cup A,S))$$

Let us take the example SFM lo illustrate how validation works. Firstly we expand the semantics of the KB so the reader can understand the validation:

$$\begin{aligned}
optional(F_{root},F_A) &\equiv state(F_A,sel) \Rightarrow state(F_{root},sel) \wedge \\
&\quad state(F_A,rem) \Rightarrow state(F_{root},rem) \\
mandatory(F_{root},F_B) &\equiv state(F_{root},sel) \Leftrightarrow state(F_B,sel) \wedge \\
&\quad state(F_{root},rem) \Leftrightarrow state(F_B,rem) \\
set_3(F_B,C_1,F_C,F_D,F_E) &\equiv state(F_B,sel) \Leftrightarrow numSelChild_3(C_1,F_C,F_D,F_E) \wedge \\
&\quad state(F_B,rem) \Rightarrow (state(F_C,rem) \wedge state(F_D,rem) \wedge state(F_E,rem)) \\
depends(F_D,F_E) &\equiv state(F_D,sel) \Rightarrow state(F_E,sel) \\
excludes(F_A,F_C) &\equiv state(F_A,sel) \Leftrightarrow state(F_C,rem) \wedge \\
&\quad state(F_C,sel) \Leftrightarrow state(F_A,rem) \\
numSelChild_3(C_1,F_C,F_D,F_E) &\equiv state(C_1,0) \Leftrightarrow (state(F_C,rem) \wedge state(F_D,rem) \wedge state(F_E,rem)) \\
&\quad state(C_1,1) \Leftrightarrow (state(F_C,sel) \oplus state(F_D,sel) \oplus state(F_E,sel)) \wedge \\
&\quad state(C_1,2) \Leftrightarrow ((state(F_C,sel) \wedge state(F_D,sel)) \vee \\
&\qquad (state(F_D,sel) \wedge state(F_E,sel)) \vee \\
&\qquad (state(F_C,sel) \wedge state(F_E,sel))) \wedge \\
&\quad state(C_1,3) \Leftrightarrow (state(F_C,sel) \wedge state(F_D,sel) \wedge state(F_E,sel)) \\
choose(F_C,sel) &\equiv state(F_C,sel) \\
choose(F_A,rem) &\equiv state(F_A,rem)
\end{aligned}$$

There exists several possible assignments for this DP. Thus, the DP is satisfiable and therefore the SFM is valid. However, if feature D were selected then $\{choose(F_D,sel)\}$

predicate would have to be added to the KB. In this case the resulting DP is unsatisfiable because the depends relationship forces to select feature E and three child features have to be selected which conflicts with the cardinality [1..2] in the set relationship. In logical terms it is justified by:

$$choose(F_D, sel), depends(F_D, F_E) \models state(F_E, sel)$$
$$choose(F_C, sel), choose(F_D, sel), state(F_E, sel), \neg state(C_1, 3), set_3(F_B, C_1, F_C, F_D, F_E) \models \bot$$

Since a valid SFM is the one that defines at least one product, this operation can be defined in terms of product listing as follows:

$$valid(SFMP(...)) \equiv products(SFMP(...)) \neq \emptyset$$

This definition of the validation operation comes from the consideration of satisfiability as a particular case of the solutions operation. Most of the existing reasoning tools, use different algorithms for satisfiability and finding all the solutions. We prefer to include the validation operation in the set of basic operations to exploit the performance issues in satisfiability algorithms.

7.2.3 Propagation

The relationships in a SFM constrain the combinations of states that are considered to be valid. It may imply that an element that apparently can have several states, can effectively have a subset of them due to existing relationships. So for example, feature B has a mandatory relationship with the Root feature. It forces feature B to have a selected state while removed state must be discarded.

User decisions have a similar impact in the states of a SFM. If a user chooses a state for an element, it may imply that some other elements are affected by this decision. So for example, if feature C is selected, feature A must be removed as a consequence.

The propagation operation receives an input SFM and obtains as a result another SFM where the conclusions that are obtained about the states of all the elements are added to the set of automatic decisions. So every combination of states and elements must be checked if it is still available or not. Figure §7.2 presents an example of propagation.

This operation can be interpreted as the extraction of conclusions from the information modelled in the DP. Since we are interested in the effect of relationships and

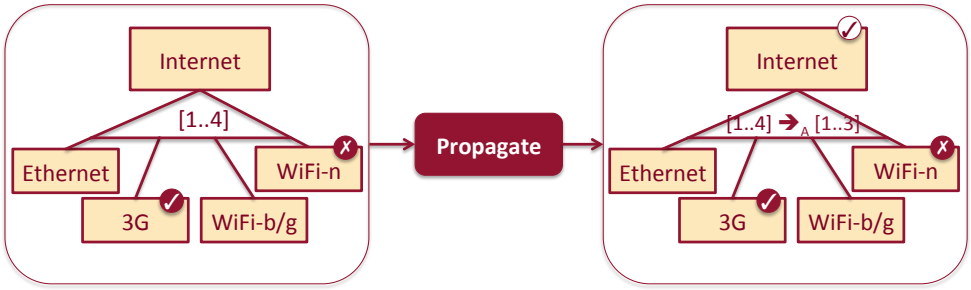


Figure 7.2: An example of a propagation operation

configurations in the state for each element in the SFM, this operation discards states that are not available anymore and sets mandatory states for each element affected by relationships and configurations. In terms of a DP, this operation checks for each element-state pair (E_i, s_j) if it can be chosen or discarded automatically. Two checks are performed for each pair:

- $isInferred(DP(...), state(E_i, s_j))$: if it is possible to infer that the element E_i must have the s_j state, then $choose(E_i, s_j)$ must be added to the set of automatic decisions.
- $isInferred(DP(...), \neg state(E_i, s_j))$: if it is possible to infer that the element E_i cannot have the s_j state anymore, then $discard(E_i, s_j)$ must be added to the set of automatic decisions

In the example, if we ignore the automatic decision that removes feature A, it can be inferred by means of propagation. Feature A cannot be selected because it is involved in an excludes relationship with feature C that is selected in the configuration:

$$
\begin{aligned}
&isInferred\ (DP(C, V, D \cup R \cup U \cup A, S), state(F_A, sel)) \text{ is true since} \\
&\quad \{choose(F_C, sel), excludes(F_A, F_C), state(F_A, sel) \oplus state(F_A, rem), ...\} \\
&\quad \equiv \{state(F_C, sel), state(F_C, sel) \Leftrightarrow state(F_A, rem) \wedge ..., state(F_A, sel) \oplus state(F_A, rem), ...\} \\
&\quad \models state(F_A, rem) \wedge \neg state(F_A, sel)
\end{aligned}
$$

Cardinalities are checked in the same way features are, it means checking each cardinal for being valid. So if feature E were selected in a configuration then cardinal 1 is discarded because:

$$
\begin{aligned}
&\{choose(F_C, sel), choose(F_E, sel), set_3(F_B, C_1, F_C, F_D, F_E), \\
&\quad (state(C_1, 1) \oplus state(C_1, 2)) \wedge state(C_1, 3), ...\} \models \neg state(C_1, 1)
\end{aligned}
$$

Repeating this inference procedure for any pair of elements and states a propagated SFM is obtained as a result. This operation can be defined as follows:

$$propagate\ (SFMP(C,V,D,R,U,A,S)) \equiv SFMP(C,V,D,R,U,A \cup Auto,S) \text{ s.t.}$$
$$Auto = \{choose(E_i,s_j)|E_i \in C,s_j \in V,isInferred(DP(C,V,D \cup R \cup U \cup A,S),state(E_i,s_j))\} \cup$$
$$\{discard(E_i,s_j)|E_i \in C,s_j \in V,isInferred(DP(C,V,D \cup R \cup U \cup A,S),\neg state(E_i,s_j))\}$$

As for product listing, the traceability table helps to identify the elements and states in the *state* predicates with the corresponding elements and states in the SFM.

7.3 SUMMARY

In this Section we have proposed a straightforward map from a SFMP to a DP. We have identified three basic query operations that are interpreted as deductive operations: product listing, validation and propagation. They can be later combined with model edition operations to perform more complex analysis operations such as errors detection or metric calculation as shown in Chapter §9.

We have set the basis to use logical reasoners to execute these operations. The use of FOL as the underlying logic of DPs helps to implement our proposal in terms of different solvers that support deductive reasoning. This benefits are studied in depth in Annex §B.

8

Interpreting explanatory operations as abduction operations

He that will not reason, is a bigot; he, who cannot, is a fool; and he, who dares not, is a slave.

Sir William Drummond (1585–1649),
Poet

In this Chapter we give a semantics for SFMs in terms of APs, which allows the use of abductive reasoning to perform some AASFM operations. Specifically, two basic explanatory operations are proposed based on the abductive operations that are useful from the AASFM point of view. In Section §8.1 we establish a connection between the AASFM and abductive reasoning. Section §8.2 proposes a mapping from SFMPs to APs that allows to perform abductive operations. Section §8.3 proposes two basic explanatory operations that cover all the explanatory analysis scenarios for the AASFM. Section §8.4 indicates how to use the traceability tables to map the results obtained for APs back to SFM artefacts. Section §8.5 proofs that some operations that were proposed for the AAFM make no sense or can be defined in terms of the proposed basic explanatory operations. Lastly, Section §8.6 summarises the contributions of this Chapter and discusses the benefits that arise from our approach.

8.1 INTRODUCTION

The goal of explanatory operations is obtaining explanations why the user decisions or the relationships cause a certain behaviour. Query operations allow the obtention of implicit data within a SFM. However, this reasoning approach is not suitable for those cases in which we want to obtain explanations why certain behaviours are given. For example, the validation operation can detect if a SFM is invalid but it provides no further information about the reasons why it is invalid. Maybe because of some conflicting relationships that avoid to find a product; or because of user decisions that violate existing relationships. In any case, deduction cannot be used to reason about the source of invalidness.

As one of the conjectures in our dissertation, we propose the interpretation of explanatory operations as abductive operations. We already proposed this approach for FMs in [85] and we propose that this hypothesis is also valid for SFMs. SFMs need to be interpreted in terms of an AP to perform any abductive operation. An AP defines a set of hypotheses that can be used to explain a given observation. Depending on the terms in which we want to obtain explanations, relationships or user decisions, two different representations of an SFM as an AP can be obtained: one that allows to obtain explanations in terms of the relationships (AP_R) and another one in terms of user decisions (AP_U).

In this Chapter we present how abductive operations can be used to define and formalise the basic explanatory operations for the AASFM. With this approach we achieve two goals: *i*) Proposing two explanatory operations to perform any explanatory analysis of SFMs *ii*) Demonstrating that a subset of explanatory operations that were defined for the AAFM in [85] makes no sense.

8.2 SFMS AS ABDUCTION PROBLEMS

In order to perform explanatory operations, an SFM is interpreted as an AP. The process to obtain it is similar to the interpretation of SFMs as DPs. Firstly, an SFM is interpreted as an SFMP. Then, the SFMP is transformed into an AP in a form that depends on the kinds of explanations to obtain. So whether explanations must be given in terms of relationships or user decisions, a different AP is used as shown in the following Sections.

8.2.1 Explaining relationships

There are many scenarios where it is important to obtain explanations in terms of relationships. For example, an invalid SFM can be detected by using the validation operation, but it is necessary to find explanations for the invalidness. These explanations are given in terms of the possible sets of conflicting relationships, which can assist the reparation process of the SFM. Another scenario is a SFM such that user decisions are set for every element in a SFM, defining a product. If the SFM is invalid and we certainly know that the user decisions define a valid configuration, it is important to obtain explanations about the invalidness in terms of the set of relationships that make it impossible to obtain that product.

Independently of the particular analysis operation, it is possible to define an AP whose objective is obtaining explanations in terms of relationships. We have to determine which logical sentences from a SFMP are the facts, the hypotheses and the observations. The first point is that the set of hypotheses must be correspond to the set of sentences that describe the relationships. This is because explanations will be defined as subsets of relationships. The set of facts defines the set of statements that are known to be true under any circumstance. It corresponds to the set of domain sentences. We certainly know that a feature can be either selected or removed and this fact is invariant and cannot be used to explain an invalid SFM. The set of observations corresponds to the set of user decisions. Therefore the objective of the AP is to explain why a certain set of decisions is valid or invalid for the given relationships. So for example, if it is not possible to select a certain feature (observation), we want to explain why it happens from the given set of relationships (hypotheses). So the AP that can be used to obtain explanations in terms of relationships results as follows[1]:

$$SFMP(C,V,D,R,U,A,S) \mapsto AP_R(C,V,(D,R,U),S,Min_S)$$

Min_S is any minimality criterion (see Section §3.2.1) to be used to select the most probable explanations. Note that the set of automatic decisions is left out of the AP. Automatic decisions are set by the propagation operation as conclusions that are obtained from the domain, relationships and user decisions. Since the relationship sentences form the set of hypotheses, the subset of intervening relationships changes from explanation to explanation, so an automatic decision could not be concluded using the subset of relationships in an explanation. Thus, automatic decisions are left out of

[1] see Section §3.2 for a description of the APs structure

the AP to avoid wrong results in reasoning. Table §8.1 shows an example of a SFM interpreted as an AP for explanatory analysis of relationships.

Abduction problem for relationships explanation
Constants (C)
$F_{root}, F_A, F_B, F_C, F_D, F_E, C_1, sel, rem, 1, 2, 3$
Variables (V)
$s_{F_{root}}, s_{F_A}, s_{F_B}, s_{F_C}, s_{F_D}, s_{F_E}, s_{C_1}$
Facts (D)
$\{state(F_{root}, sel),$ $state(F_A, sel) \oplus state(F_A, rem),$ $state(F_B, sel) \oplus state(F_B, rem),$ $state(F_C, sel) \oplus state(F_C, rem),$ $state(F_D, sel) \oplus state(F_D, rem),$ $state(F_E, sel) \oplus state(F_E, rem),$ $(state(C_1, 1) \oplus state(C_1, 2)) \wedge \neg state(C_1, 3),$ $state(F_{Root}, s_{F_{Root}}) \wedge state(F_A, s_{F_A}) \wedge state(F_B, s_{F_B}) \wedge$ $state(F_C, s_{F_C}) \wedge state(F_D, s_{F_D}) \wedge state(F_E, s_{F_E}) \wedge state(C_1, s_{C_1})\}$
Hypotheses (R)
$\{optional(F_{root}, F_A),$ $depends(F_D, F_E),$ $mandatory(F_{root}, F_B),$ $excludes(F_A, F_C),$ $set_3(F_B, C_1, F_C, F_D, F_E)\}$
Observations (U)
$\{choose(F_C, sel), choose(F_D, sel)\}$
Minimality criterion
$min_S(E) \equiv \{e \mid e \in E \wedge \neg\exists e'(e' \in E \wedge e' \subset e)\}$

Table 8.1: Abduction problem for relationships explanation

8.2.2 Explaining configurations

Explanations can also be obtained in terms of user decisions. For example, a SFM can be invalid due to conflicting user decisions. The conflict can be caused by decisions that violate some relationships or contradictory decisions made by different users such as the selection and removal of the same feature. In any case, an explanation is a subset of user decisions that must be corrected in order to obtain a valid SFM.

Defining an AP to obtain user decisions as explanations is quite similar to the previous approach. In this case, the set of hypotheses corresponds to the set of configuration sentences since they are used to obtained explanations. A first temptation is setting the set of relationships as observations in symmetry with SFM explanatory operations. However, the set of facts that is certainly known to be true is not only comprised of domain sentences, but relationship and automatic decision sentences. The relationships are known to be correct so they are considered as facts. Thus, the set of observations is empty. This happens because it is not possible to infer from just one configuration all the relationships in a SFM such as a mandatory relationship between two features or a set relationship among a set of features. Therefore an AP can be obtained from a SFMP as follows:

$$SFMP(C,V,D,R,U,A) \mapsto AP_C(C,V,(D \cup R,U,\emptyset),S,Min_S)$$

The automatic decisions are also left out of the AP. All the automatic decisions inferred from the set of relationships and the domain can be kept without any problem since the formulas in $D \cup R$ are facts, therefore $D \cup R \models A$. However automatic decisions inferred from user decisions, i.e. $D \cup R \cup U \models A$ must be left out of the AP since no predicate in U is certainly known to be true or false.

As for explaining relationships, a minimality criterion is added. Table §8.2 shows an example of an AP that represents an example SFM.

8.3 BASIC EXPLANATORY OPERATIONS

Abductive operations, specifically minimal explanations and minimal conflict sets, can be used to perform abduction on the two APs obtained to obtain explanations in terms of relationships and decisions. The prerequisite of minimal explanations demands for a non-empty set of observations. The prerequisite of minimal conflict sets

Abduction Problem for configuration explanation
Constants (C)
$F_{root}, F_A, F_B, F_C, F_D, F_E, C_1, sel, rem, 1, 2, 3$
Variables (V)
$s_{F_{root}}, s_{F_A}, s_{F_B}, s_{F_C}, s_{F_D}, s_{F_E}, s_{C_1}$
Facts ($D \cup R$)
$state(F_{root}, sel)$
$state(F_A, sel) \oplus state(F_A, rem)$
$state(F_B, sel) \oplus state(F_B, rem)$
$state(F_C, sel) \oplus state(F_C, rem)$
$state(F_D, sel) \oplus state(F_D, rem)$
$state(F_E, sel) \oplus state(F_E, rem)$
$(state(C_1, 1) \oplus state(C_1, 2)) \wedge \neg state(C_1, 3)$
$state(F_{Root}, s_{F_{Root}}) \wedge state(F_A, s_{F_A}) \wedge state(F_B, s_{F_B}) \wedge$ $state(F_C, s_{F_C}) \wedge state(F_D, s_{F_D}) \wedge state(F_E, s_{F_E}) \wedge state(C_1, s_{C_1})$
$optional(F_{root}, F_A)$
$depends(F_D, F_E)$
$mandatory(F_{root}, F_B)$
$excludes(F_A, F_C)$
$set_3(F_B, C_1, F_C, F_D, F_E)$
Hypotheses (U)
$choose(F_C, sel)$
$choose(F_D, sel)$
$choose(F_E, sel)$
Minimality criterion
$min_S(E) \equiv \{e \mid e \in E \wedge \neg \exists e' (e' \in E \wedge e' \subset e)\}$

Table 8.2: Abduction Problem for configuration explanation

demands for an empty set of observations. Therefore, only minimal explanations can be used for the AP_R and minimal conflict sets for the AP_C.

Trinidad and Ruiz-Cortés [85] propose a set of scenarios where explanations are relevant for FMs. These scenarios can be easily adapted to SFMs. All these scenarios can be categorised in two cases: the SFM under analysis is valid and the SFM is invalid.

If we apply this categorisation for the two abductive operations that can be used for SFMs, four combinations arise: obtaining minimal explanations for a valid and invalid SFM described as a AP_R; and obtaining the minimal conflict sets for a valid and invalid SFM described as a AP_C. In Section §8.5 we demonstrate that obtaining explanations from valid SFMs makes no sense. The two remaining explanatory operations, *'why are the relationships not valid?'* and *'why are the user decisions not valid?'*, can be interpreted in terms of AP operations as we describe next.

8.3.1 Why are the relationships not valid?

This operations explains why a given SFM is invalid in terms of the relationships that cause it. Relationships are considered as the source of invalidness while user decisions in a configuration are considered to be correct. Several explanations can be obtained and it is responsibility of the user to determine which is the adequate explanation and the changes to realise in the set of relationships in order to repair the SFM. Figure §8.1 shows an example for this operation.

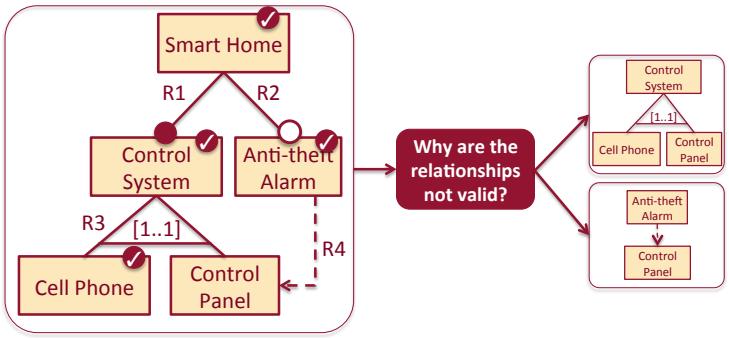


Figure 8.1: An example of 'why are the explanation not valid? operation

If a SFM is invalid, then the corresponding $DP(C,V,KB,S)$ is insatisfiable and therefore $KB = D \cup R \cup U$ is inconsistent. As for any two sets of logical sentences A and B, $A \cup B$ is inconsistent iff $A \models \neg B$. We can affirm that $D \cup R \cup U$ is inconsistent iff $D \cup R \models \neg U$.

For example, if after a configuration $U = \{choose(F,sel)\}$ the resulting SFM is not valid then $D \cup R \cup \{choose(F,sel)\}$ is inconsistent. Thus, We can affirm that $D \cup R \models \{\neg choose(F,sel)\}$, i.e. the domains and relationships impedes feature F to be selected by a user. In case we are interested in obtaining the subsets of relationships ($\Delta \subseteq R$) that impede the selection of feature F, we can obtain all the minimal explanations as

follows:

$$MinExpl(AP_C(C,V,(D,R,\{\neg choose(F,sel)\}),S,Min)$$
$$\equiv Min(\{\Delta|\Delta\subseteq R, D\not\models\{\neg choose(F,sel)\}, D\cup\Delta\models\{\neg choose(F,sel)\}\})$$

This way all the minimal subsets of hypotheses from which $\{\neg choose(F,sel)\}$ can be concluded are obtained.

Therefore, if we want to determine why a SFM is not valid in terms of its relationships, we have to explain why the negation of the user decisions can be concluded from the set of relationships and domain. Let us consider the example in Table §8.1. We want to explain which relationships impede the selection of features `C` and `D` at the same time. To obtain explanations we have to explain why the negation of the configuration can be concluded from the domain and relationships, i.e. $\{\neg choose(F_C,sel)\vee\neg choose(F_D,sel)\}$ can be concluded. To obtain explanations for this case, an AP is solved obtaining the following result:

$$\begin{aligned}
&MinExpl(AP_C(C,V,(D,R,\{\neg choose(F_C,sel)\vee\neg choose(F_D,sel)\}),S,Min))\\
&\equiv\{\Delta|\Delta\subseteq R,\quad D\not\models\{\neg choose(F_C,sel)\vee\neg choose(F_D,sel)\}\},\\
&\qquad\qquad D\cup\Delta\models\{\neg choose(F_C,sel)\vee\neg choose(F_D,sel)\}\}\\
&=\{\quad\{depends(F_D,F_E)\},\{set_3(F_B,C_1,F_C,F_D,F_E)\},\\
&\qquad\{depends(F_D,F_E),set_3(F_B,C_1,F_C,F_D,F_E)\},...\}
\end{aligned}$$

From all the explanations, we are not interested in all of them, but in the minimal ones. Which minimality criterion fits into our problem must be determined depending on the nature of the problem to solve. For example, the minimal subset criterion produces two explanations $\{depends(F_D,F_E)\}$ and $\{set_3(F_B,C_1,F_C,F_D,F_E)\}$ since all the remaining explanations are supersets of it. It means that either the depends or the set relationships must be changed in order to become the configuration valid. At this point, the use of relationship predicates instead of *state* predicates helps to apply the minimality criterion proportionally to the number of relationships. This way, we ensure that the relationship is not divided in smaller logical sentences for the explanatory analysis.

The minimal number of assumptions can be also used to weight those explanations with the minimum number of relationships. The selection of the most suitable minimality criterion for the AASFM is out of the scope of this dissertation and should be the objective of future works.

As a conclusion, the minimal explanation operation is suitable to solve the *'why are the relationships not valid?'* operation, that can be defined as follows:

$$WhyNotRels(SFMP(C,V,D,R,U,A,S)) \equiv MinExpl(AP_C(C,V,(D,R,\neg U),S,min_S))$$

For each particular case of explanation, we verbalise the observation obtaining a variety of SFM explanatory questions or *explanatory scenarios*. So for the invalid configuration $\{choose(F,sel)\}$, this operation can be verbalised as 'why is it not valid to select feature F?'. Since this expression can be very verbose, some equivalent questions are usually used such as 'why is feature F dead?'. Since this question does not use a negation in its expression it may conduce to misleadings and interpret it as a particular case of 'why is a SFM valid given a configuration?'. Section §9.4 presents a catalog of explanatory operations where most used explanatory operations are verbalised and bound to specific configurations.

8.3.2 Why are the user decisions not valid?

This operation obtains explanations in terms of the user decisions that make a SFM invalid. This operation considers that the relationships are correct while it is the configuration what produces the invalidness. Figure §8.2 shows an example for this operation.

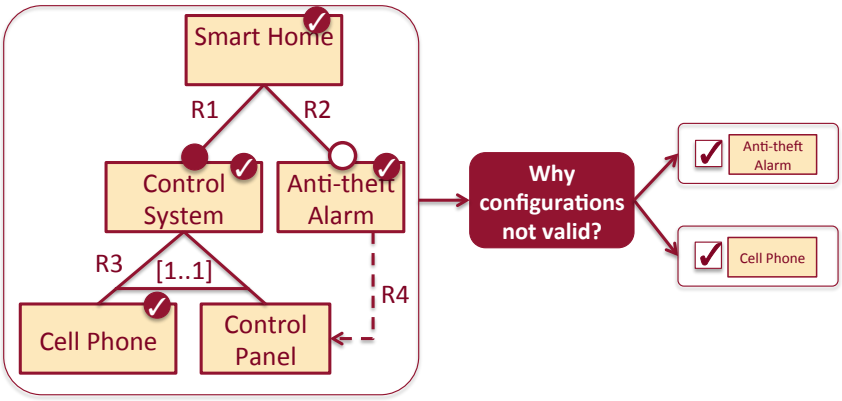


Figure 8.2: An example of 'why is a configuration not valid?' operation

A SFM is invalid if the $DP(C,V,KB,S)$ that describes the model has no solution. It means that $D \cup R \cup U$ is inconsistent. The objective of this operation is determining which subsets of the configuration ($\Delta \subseteq U$) generates such inconsistency. The relationships and domain must be valid as a precondition so that $D \cup R$ is consistent. The AP

that describes this problem has an empty set of observations so the minimal explanation operation cannot be used to obtain explanations. In this case, the minimal conflict set operation comes into play providing Δ sets that are inconsistent with the set of facts;

$$\begin{aligned} &MinConflicts(AP_R(C,V,(D \cup R,U,\emptyset),S,min)) \\ &\equiv min(\{\Delta | \Delta \subseteq U, D \cup R \text{ is consistent}, D \cup R \cup \Delta \models \bot\}) \end{aligned}$$

Note that the precondition that $D \cup R$ must be consistent, establishes that the SFM must be valid without any configuration. For the example SFM in Table §8.2, features `C`, `D` and `E` are selected by user decisions. The SFM without any configuration is valid since it defines several products. The configuration makes an SFM invalid since the `<1..2>` cardinality of the set relationship is violated. Each subset of the configuration is checked if it is a conflict set obtaining the following results:

$D \cup R \cup \{choose(F_C,sel),choose(F_D,sel),choose(F_E,sel)\}$ is inconsistent
$D \cup R \cup \{choose(F_C,sel),choose(F_D,sel)\}$ is inconsistent
$D \cup R \cup \{choose(F_C,sel),choose(F_E,sel)\}$ is consistent
$D \cup R \cup \{choose(F_D,sel),choose(F_E,sel)\}$ is consistent
$D \cup R \cup \{choose(F_C,sel)\}$ is consistent
$D \cup R \cup \{choose(F_D,sel)\}$ is consistent
$D \cup R \cup \{choose(F_E,sel)\}$ is consistent

Thus two conflict sets are obtained: $\{choose(F_C,sel),choose(F_D,sel),choose(F_E,sel)\}$ and $\{choose(F_C,sel),choose(F_D,sel)\}$. Since the first one is a superset of the second one, the second one is a minimal conflict set.

We define the *'Why are the user decisions not valid?'* operation as the realisation of a minimal conflict set operation on an AP for configuration explanation as follows:

$$WhyNotConf(SFMP(C,V,D,R,U,A,S)) \equiv MinConflicts(AP_R(C,V,(D \cup R,U,\emptyset),S,min))$$

8.4 TRACEABILITY

8.4.1 Relationships explanations

The traceability table assists the mapping from AP results to relationships in the original SFM. There is an indirect one-to-one relationship between predicates in the

hypotheses set and relationships. Each predicate indicates the kind of relationship, and its constants represent the elements in the SFM that are affected by the relationship. Since a SFM can only contain one relationship of the same kind that affects a set of elements, it is quite easy to obtain the relationship that corresponds to a predicate.

So for the example, we obtained $\{depends(F_D, F_E)\}$ as an explanation. The constants F_D and F_E correspond to features `D` and `E` respectively. Thus, the predicate in the explanation represents a *depends* relationship between features `D` and `E` unambiguously.

8.4.2 Configuration explanations

The explanations for a 'why is a configuration not valid?' operation are *choose* and *discard* predicates. They can be mapped back to the corresponding `ChooseStateConstraint` or `DiscardStateConstraint` instances in the SFM if there exist only one instance that affects a specific pair of element and state. For the previous example, the *choose* predicates can be traced to the SFM to determine that selecting `C` and `D` at the same time is a source of conflict. Thus, either the `ChooseStateConstraint` instance that selects `C` or the one that selects `D` must be removed from the SFM to solve the conflict.

Note that there exists no problem in case there exist more than one user decision selecting or discarding a state. In that case, all the user decisions that refer to the same pair of element and state will be in the explanation at the same time.

8.5 DISCARDING SENSELESS EXPLANATORY SCENARIOS

In this Section we present some scenarios where abductive reasoning cannot be applied to analyse SFMs. Firstly, we present the kinds of configuration that lead to no explanation at all. Secondly, we present two 'why?' operations that were proposed for FMs in [85] as an exploitation of abductive reasoning. We rely on the formalisation of the explanatory analysis to discard 'why?' operations as an alternative explanatory operation.

8.5.1 Invalid configurations

In a *WhyNotRels* operation it is not possible to obtain explanations if the precondition $D \not\models U$ for the minimal explanation operation is violated. There are only two cases where a SFM violates this precondition:

- Root feature is removed: The configuration that produces inconsistency contains the $\{choose(F_{Root}, rem)\}$ sentence. So the *WhyNotRels* operation searches for the minimal explanations for $\{\neg choose(F_{Root}, rem)\}$. Since the domain sentences forces the root feature to be selected, the precondition of minimal explanation operation is violated, because the observation can be inferred from the domain i.e. $state(F_{Root}, sel) \models \{choose(F_{Root}, rem)\}$.

- Unique cardinal is chosen: A cardinality C_i in a set relationship has a unique valid cardinal n. So from the domain it can be inferred that $D \models \{state(C, i, n)\}$. If a configuration sets the sentence $\{choose(C_i, n)\}$ then a *WhyNotRels* operation searches for explanations for $\{\neg choose(C_i, n)\}$ which clearly contradicts the domain sentences, so the precondition is violated.

As a conclusion, no explanation can obtained if a configuration contradicts any domain sentence.

8.5.2 Why are the relationships valid?

In [85], the so-called 'why?' operations were proposed for the AAFM. They were used to obtain explanations when a FM is valid, instead of when it is invalid. In this Section we rely on the previous formalisation of explanatory operations to demonstrate that these operations for SFMs either makes no sense from the analysis point of view or can be defined in terms of a 'why are the relationships not valid?' operation.

A SFM is valid if the corresponding $DP(C, V, KB, S)$ is consistent. It means that $D \cup R \cup U$ is consistent. The general definition of an explanation operation is:

$$MinExpl(AP(C, V, (D, R, U), S, Min) \equiv Min(\{\Delta | \Delta \subseteq R, D \not\models U, D \cup \Delta \models U\})$$

Analysing the above definition of explanation there are two possible situations for this operation if $D \cup R \cup U$ is consistent:

1. Explanations can be obtained if there exists at least a subset Δ of R such that $D \cup \Delta \models U$. Since $D \cup R \cup U$ is consistent because the SFM is valid, for any subset $\Delta \subseteq R$, $D \cup \Delta \cup U$ must be consistent. At least an explanation Δ is obtained if $D \cup \Delta \models U$. Since $D \cup \Delta \cup U$ and $D \cup R \cup U$ are consistent we can affirm that:

$$D \cup \Delta \models U, \Delta \subseteq R, D \cup R \cup U \text{ is consistent} \Rightarrow D \cup R \models U.$$

If $D \cup R \models U$, then it can be affirmed that $D \cup R \cup \neg U$ is inconsistent. Thus we conclude that explanations are only obtained for a 'why?' operation if it really corresponds to a 'why are the relationships not valid?' for the negation of the configuration. In other words:

$$WhyRels(SFMP(C,V,D,R,U,A,S)) \equiv WhyNotRels(SFMP(C,V,D,R,\neg U,A,S))$$

2. No explanation is obtained under two circumstances:

   (a) If the precondition $D \not\models U$ is violated it is because either:

   - The observation is $\{state(F_{Root},sel)\}$ which is part of the set of facts F. The root must always be selected for any product and asking why it is so is a trivial question.
   - The observation is $F \models \{state(C,i,n)\}$ being n a unique valid cardinal for a cardinality C_i of a set relationship also contradicts the precondition.

   (b) For no subset of $\Delta \subseteq R$, U is a logical consequence. For the given example, determining why feature A is selectable makes no sense since features are modelled to be selectable. If we set the configuration $\{choose(F_A,sel)\}$, it is consistent with the existing relationships but it cannot be concluded from the set of domain and relationship sentences. In this case, the 'why?' operation obtains no result and therefore it makes no sense from the analysis point of view.

From the above analysis, we extract an important conclusion that can even be extrapolated to the AAFM:

> *why?' operations make no sense unless they are implicitly solving a 'why not?' operation.*

8.5.3 Why is a configuration valid?

In a symmetry with the previous operation, we have studied if it makes sense to perform a 'why?' operation for configurations instead of relationships. In this case, the

input SFM must be valid, therefore $D \cup R \cup U$ is consistent. Regarding the 'why not?' case, explanation problems does not fit into this case since there is no observation. Conflict sets neither fit into this kind of problem since there is no inconsistency. So what can we expect from this operation? No result at all. A configuration is expected to be valid by default, so asking why that is so is a triviality. Therefore 'why is a configuration valid given a SFM?' is an operation that makes no sense and must be removed from the catalog of explanatory operations.

8.6 SUMMARY

In this Chapter we have proposed an interpretation of SFMs as APs to perform the explanatory analysis of SFMs. Two different interpretations are given to obtain explanations in terms of relationships or configurations. For each interpretation, one basic explanatory operation is defined to explain the reasons why a SFM is invalid. So the operation 'why are the relationship not valid?' explains an invalid SFM in terms of relationships and 'Why are the user decisions not valid?' explains an invalid SFM in terms of user decisions. These operations covers all the possible analysis scenarios that have been proposed at date for FMs as it is shown in Chapter §9.

As an important conclusion we have not only proposed two explanatory operations but we have demonstrated that the so-called 'why?' operations for the explanatory analysis of FMs can either be defined in terms of a 'why are the relationship not valid?' operation or they make no sense from the analysis point of view.

At this point, we have proposed five basic operations for the AASFM. In the next Chapter we present how they can be combined to perform more complex analysis operations that covers most of the user needs.

9

A Catalogue of Analysis Operations

I often liken the process of physics research to solving a jigsaw puzzle. As we put together pieces to form patches, a certain image of the overall picture emerges, but until the game is sufficiently progressed, we are not quite sure.

Benjamin W. Lee (1935–1977),
Theoretical physicist

Basic operations, either query or explanatory, have set the basis of the AASFM. In this Chapter we explore how these basic operations can be combined to support the automated analysis. For that sake, section §9.1 presents a categorisation of operations. Section §9.2 presents a summary on the basic operations proposed in previous Chapters. Section §9.3 presents a wide catalogue of compound query operations. Section §9.4 presents a set of configurations or scenarios that can be used to realise different compound explanatory operations for SFMs. Section §9.5 shows some examples where query and explanatory operations are combined to perform compound mixed operations. Last, in Section §9.6, some conclusions are approached to study the application of the proposed catalogue of AASFM operations.

9.1 INTRODUCTION

In order to define a catalogue of AASFM operations as complete as possible, we take the most complete and thorough catalogue proposed to date by Benavides et al. [14]. However, we want to go further than a one to one correspondence of operations. We propose a new classification of analysis operations that defines a reduced set of operations sufficient to perform the AASFM. It will allow to define new operations as a composition of them and to propose implementations of the AASFM that are fully-fledged with few operations. We distinguish the following kinds of AASFM operations:

- Basic: cannot be defined in terms of other operations. Depending on the kind of reasoning that is used to solve them, two subtypes of basic operations are proposed:
  - Explanatory operations: The goal of explanatory operations is obtaining explanations why the configuration or the relationships are provoke a certain behaviour. They can be interpreted as abduction problems.
  - Query operations: the goal of a query operation is to extract implicit information from a SFM and make it explicit. Explanatory operations could be considered to fit into this definition so a query operation can also be defined as any non-explanatory operation. They can be interpreted as deduction problems.
- Compound: a compound operation can be defined on top of model edition and basic operations. If a compound operation only combines basic query operations, it is called a compound query operation. Symmetrically, if a compound operation only relies on basic explanatory operations, it is called a compound explanatory operation. In case a compound operation combines query and explanatory operations it is called a compound mixed operation.

In this Chapter we present how to use compose model edition and basic operations to perform compound operations. The possible combinations grow exponentially being unfeasible to present a complete catalogue of operations. So in this Chapter we present how the most used operations for the AAFM are reinterpreted as compound operations for the AASFM. This approach increases the added-value of the AASFM whose reasoning capabilities increase significantly compared with the AAFM.

Basic Operations		
SFM Operation	**Problem**	**Reasoning Operation**
Product Listing	DP	Solutions
Propagation	DP	Inference
Validation	DP	Satisfiability
Why are the relationships not valid?	AP_R	Minimal Explanations
Why is a configuration not valid?	AP_C	Minimal Conflict Set

Table 9.1: Basic operations in the AASFM

9.2 BASIC OPERATIONS

Five basic operations, that are summarised in Table §9.1, have been defined in previous Chapters for different analysis purposes. We choose to Java as a language to describe how basic and model edition operations are combined. Firstly, it provides a syntax to manipulate metamodel instances. Secondly, model edition operations have already been defined as methods in the SFMM. Thus, each basic operation is assigned to a method prototype in Java that provides its functionality:

- **Product listing**: The prototype for this operation is `Set<Product> products(SFM inputSFM)` such that the `Product` class encapsulates a `Map<Element,State>` instance that contains element-state pairs.
- **Validation**: The prototype for this operation is `boolean valid(SFM input)`.
- **Propagation**: The prototype for this operation is `SFM propagate(SFM input)`.
- **Why are the relationships not valid?**: The prototype for this operation is `Set< Set <Relationship>> whyNotRels (SFM input)`. An explanation is given in the form `<Set<Relationship>`. Since several explanations can be obtained, the obtained result is a set of explanations `Set<Set<Relationship>>`.
- **Why is the configuration not valid?**: The prototype for this operation is `Set< Set< Configuration>> whyNotConf (SFM input)`. As it is obvious the configuration in the SFM cannot be empty, otherwise there would be no configuration to diagnose.

The implementation of these methods will be discussed in Annex §B.

9.3 COMPOUND QUERY OPERATIONS

A compound query operation can be defined as a composition of model edition and basic query operations. Three model edition methods are used for the definitions: `setUserDecisions`, `setAutoDecisions` and `reset`.

9.3.1 Void SFM

A SFM is said to be void if it defines no product at all independently of the configurations that are set. A SFM is void mainly due to contradictory relationships that cannot be satisfied at the same time. To ensure that the SFM contains no configuration constraint, the SFM must be reset prior to validation. This operation can be defined as follows:

```
boolean voidSFM (SFM inputSfm) {
        SFM resetSfm = reset(inputSfm);
        return !valid(resetSfm);
}
```

Figure §9.1 shows an example of this operation. Note the difference between an invalid and a void SFM. A SFM is invalid if its configuration does not satisfy all the relationships in the model. A SFM is void if after removing any configuration the SFM is still invalid. Thus, a void SFM is invalid for any given configuration.

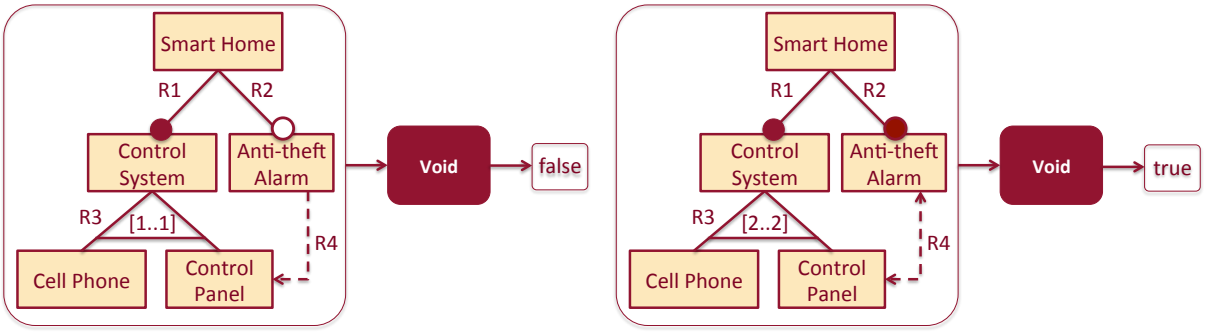


Figure 9.1: An example of a void and a non-void (valid) SFM

9.3.2 Products counting

This operation counts the number of different products a SFM describes for the given configuration. This operation can be seen as the cardinal of products obtained by the products listing operation so it can be defined in the following form:

```
int count (SFM inputSFM) {
        return inputSFM.products.size();
}
```

For an invalid SFM, this operation counts zero products. An example of this operation is shown in Figure §9.2. For the invalid SFM in Figure §9.1 the products counting operation returns zero.

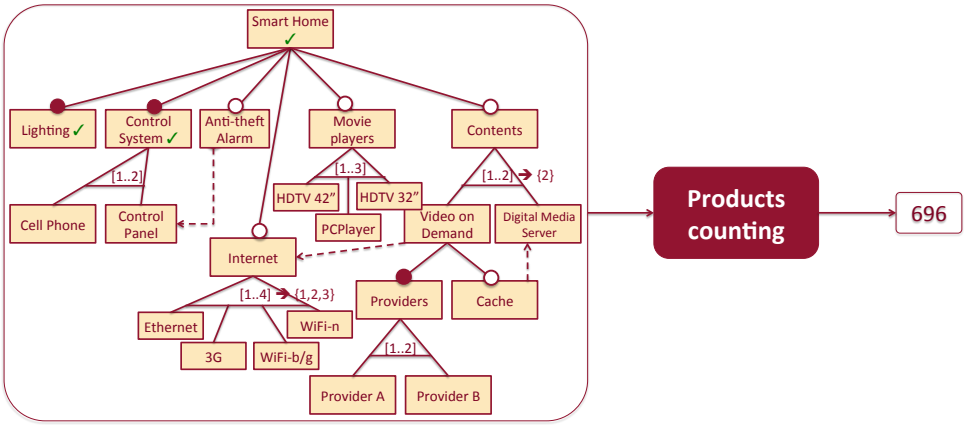


Figure 9.2: An example of products counting

9.3.3 Configure with propagation

In an individual configuration process, the propagation operation can be executed just after a user decision since it is not possible to receive other configurations from other stakeholders. This operation is implemented as follows:

```
SFM configWithPropagation(SFM inputSfm, Set<Configuration> conf) {
        SFM intermediateSfm = setUserDecisions(inputSfm, conf);
        SFM outputSfm = propagate(intermediateSfm);
        return outputSfm;
}
```

Figure §9.3 shows an example for this operation.

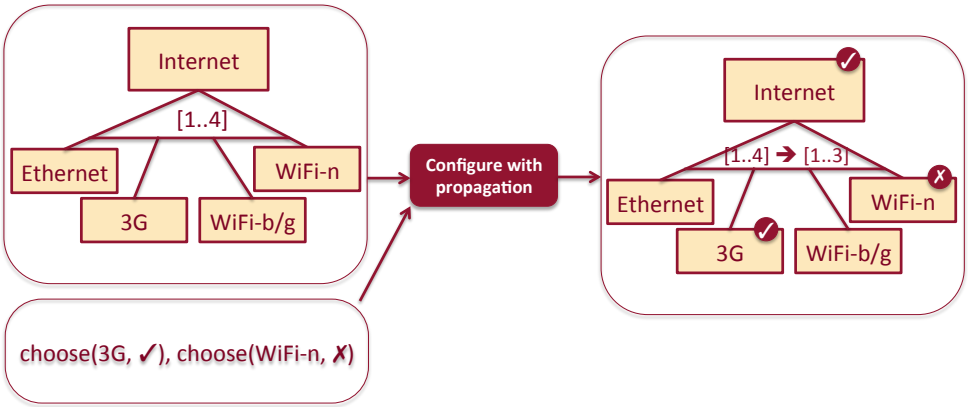


Figure 9.3: An example of configure with propagation

9.3.4 Filtering

It produces a list of products that satisfies a given configuration. The SFM must be firstly reset to delete any previous configuration. Then the input configuration is set and a propagation is performed. The resulting SFM is used to obtain a set of products satisfying the given input configuration:

```
Set<Product> filter(SFM inputSfm, Set<Configuration> conf) {
        SFM resetSfm = reset(inputSfm);
        SFM propagatedSfm = configWithPropagation(resetSfm,conf);
        return products(propagatedSfm);
}
```

Figure §9.4 shows an example for this operation.

9.3.5 Optimisation

An optimisation operation orders the set of products obtained in a product listing operation following a certain criterion. The result is a totally ordered set $O = (P, \leq)$ whose order is determined by an order relation $\leq$ that defines the order criterion. A total order implies the definition of an order for any pair of products in a SFM. An example of total order relation is the one that orders the products in terms of the number of selected features. It is applied to the SFM in Figure §9.5 to obtain the products with the minimal number of features.

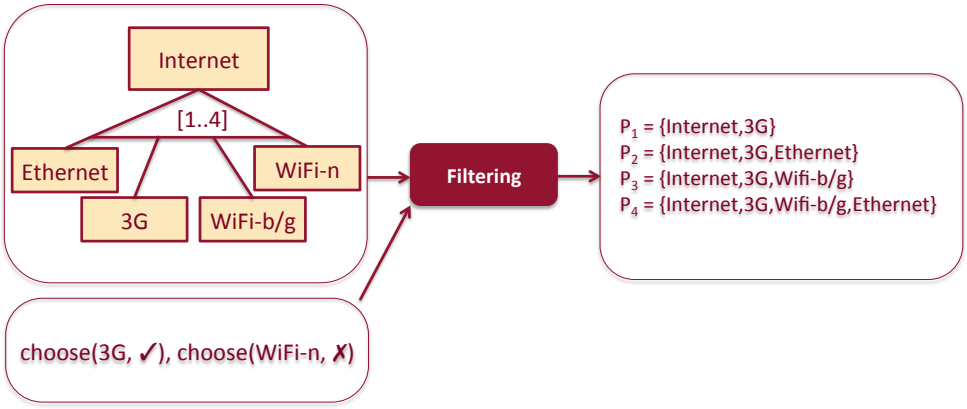


Figure 9.4: An example of filtering

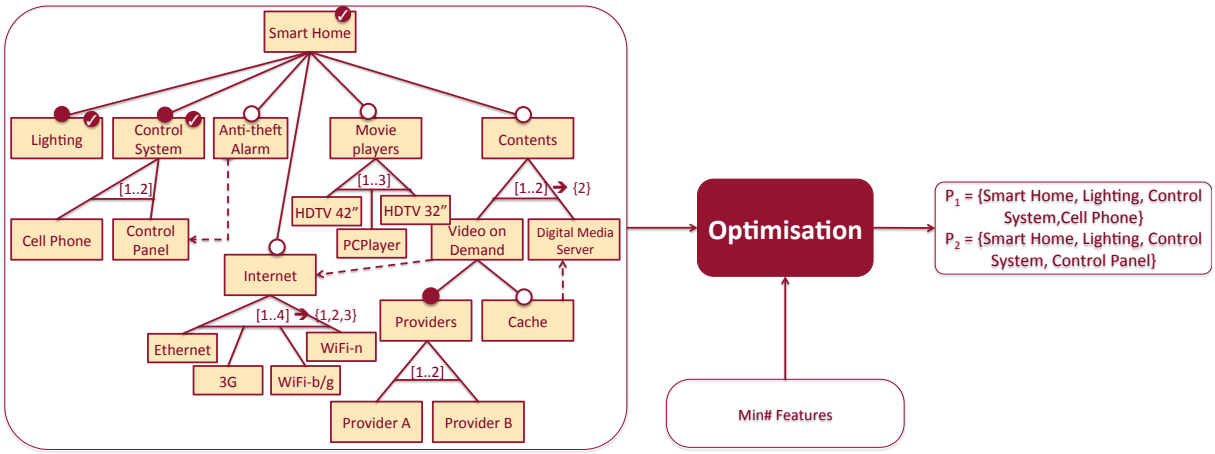


Figure 9.5: An example of optimisation operation

This operation firstly obtains the unordered set of products of the input SFM and then it is ordered following a criterion:

```
SortedSet<Product> optimise(SFM inputSfm, Comparator<Product> criterion) {
        // any other implementation of SortedSet can be used instead.
        SortedSet<Product> output = new TreeSet<Product>(criterion)
        result.addAll(productsListing(inputSfm));
        return output;
}
```

The optimisation criterion is described as an order relationship defined by a comparator. Comparators define a total ordering between any pair of objects, in our case products. Comparators simplify the implementation of optimisations.

9.3.6 Anomalies detection

The void SFM operation detects if a SFM at least defines one product or not. However, it cannot ensure that the SFM defines all the products that can be built in a SPL. To ensure this, we need a complete list of products to check that a SFM defines exactly that intended list. It is possible for small SPLs but unfeasible for large-scale SPLs. In this case, anomalies are used to detect symptoms of mismodelling that suggest a possible failure in a SFM definition. So anomalies detection is a basic activity in the quality assurance of SFMs. We define four anomalies detection operations, one for each kind of anomaly:

- **Dead Features**: a feature is dead if it cannot be selected in any product and therefore appears in no product at all. A Dead feature can be detected if it is the only selected feature in the SFM and the model is invalid. In other words, it is not possible to select the feature under any circumstances. Note that all the anomalies detection operations need to reset the SFM to perform the analysis. This operation can be defined in terms of other analysis operation as follows:

```
boolean isDead(SFM inputSfm,GenericFeature f) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        configSet.add( new ChosenElementConstraint
                                    (f,new SelectedState()) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm,configSet);
        return !valid(configuredSfm);
}
```

- **False-Optional feature**: A feature is optional if it is the child in a non-mandatory relationship, such as optional or set relationships. A feature is false-optional if despite of being modelled as optional, it must be selected whenever its parent feature is selected. In other words, a false-optional feature has an implicit mandatory relationship with its parent feature. The most frequent way to detect if a feature is false-optional consists of setting a configuration where the parent feature is selected and the feature under analysis is removed. If the resulting SFM is invalid then the feature is false-optional. This anomaly can be detected for a specific optional feature as follows:

```
boolean isFalseOptional(SFM inputSfm, Feature f) {
        Set<Configuration> configSet = new HashSet<Configuration>();
```

```
        configSet.add( new ChosenStateConfiguration
                              (f, new RemovedState()) );
        configSet.add( new ChosenStateConfiguration
                              (f.parent(),new SelectedState()) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm,configSet);
        return !valid(configuredSfm);
}
```

Note that the input feature is an instance of `Feature` class instead of an instance of `GenericFeature`. Since a root feature is not an optional feature we avoid receiving it as an input for this operation.

- **Wrong Cardinals**: a cardinality in a set relationship indicates the number of child features or cardinals that must be selected in every product. A cardinal is wrong if there is no product containing that number of child features. To perform this operation, the specific cardinal is chosen in a configuration. This anomaly can be detected for a given cardinal and cardinality as follows:

```
boolean isWrongCardinal(SFM inputSfm, Cardinality card,
                        Cardinal value) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        configSet.add( new ChosenStateConfiguration(card, value) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm, configSet);
        return !valid(configuredSfm);
}
```

- **Unique Cardinals**: a cardinal in a set relationship is unique if all the products only contain that number of child features, while remaining cardinals are wrong. This operation is implemented by the `isUniqueCardinal` method, which checks if it is possible to remove a cardinal:

```
boolean isUniqueCardinal(Cardinality card, Cardinal value) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        configSet.add( new DiscardElementConstraint(card,value) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm,configSet);
        return !valid(configuredSfm);
}
```

This operation has not been defined to date for FMs, so it can be considered as a novel contribution for SFMs and FMs. It is the result of the definition of frequent configurations or scenarios proposed in Section §9.4.

Figure §9.6 shows an example for each kind of anomaly. Repeating the above operations for every element in a SFM where they are applicable (features and cardinalities) is also known as *error detection* operation. Its result is a set of errors that affect a SFM.

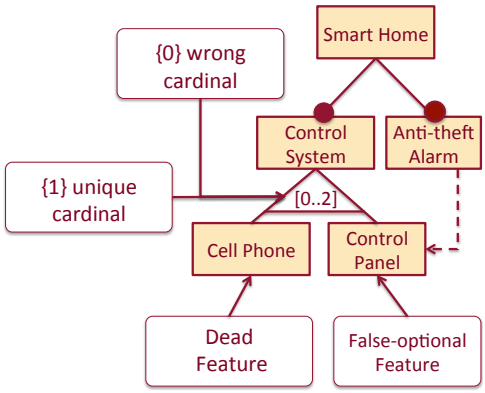


Figure 9.6: Example of anomalies in SFMs

For a complete study of the anomalies in a SFM, the above operations must be executed several times for each feature or cardinality. It is possible to define compound operations to detect all the errors of a kind. As an example, we propose an operation to detect all the dead features in a SFM:

```
public Set<Feature> detectDeadFeatures(SFM inputSfm) {
        // any other implementation of Set can be used instead.
        Set<Feature> output = new TreeSet<Feature>();
        for( f : features) {
                if (isDead(inputSfm,f))
                        output.add(f);
        }
        return output;
}
```

9.3.7 Core features

A feature is core if it appears in each and every product of a SFM. In other words, a feature is core if it cannot be removed from any product. This operation can be defined

as follows:

```
boolean isCore (SFM inputSfm, Feature f) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        configSet.add( new ChosenStateConfiguration(f,
                                                    new RemovedState ()) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm, configSet);
        return !valid(configuredSfm);
}
```

This operation can be executed for every feature in a SFM, resulting the following operation:

```
public Set<Feature> coreFeatures(SFM inputSfm) {
        Set<Feature> output = new TreeSet<Feature>();
        for( f : inputSfm.getFeatures() ) {
                if (isBasic(inputSfm, f))
                        output.add(f);
        }
        return output;
}
```

9.3.8 Variant feature

A variant feature is the one that appears in at least one product but not in all of them. It can be also defined as a non-core and non-dead feature. Using this definition, this operation is defined as follows:

```
public boolean isVariant(SFM inputSfm,Feature f) {
        return !isBasic(inputSfm,f) && !isDead(inputSfm,f);
}
```

It is possible to propose an operation to detect all the variant features in the same terms as the operation defined for core features.

9.3.9 Metrics

Metrics operations extract quantitative information from a SFM. It is not the scope of this dissertation to provide for a complete catalogue of metric operations. However

we present two of them as samples:

- **Variability Degree**: it measures how restrictive a SFM is. The most flexible SFM that can be built given a set of N features, is the one where all the features are optional, and therefore any combination of them is allowed. In this situation, the number of products that can be built with the most flexible SFM is 2^N. So for the example in Figure §9.2, the variability degree is 0,00531.This metric compares the number of products in a SFM to the most flexible SFM. Following we show an example of implementation of a method supporting this operation:

```
public double variabilityDegree(SFM inputSfm) {
        SFM resetSfm = reset(inputSfm);
        int numProducts = count(resetSfm);
        int numFeatures = inputSfm.getFeatures().size();
        int numProductsNoConstraint = Math.pow(2,numFeatures);
        return numProducts / numProductsNoConstraint;
}
```

- **Commonality Factor**: The commonality factor for a set of features indicates the ratio of products where the features in an input set are selected at the same time. To count the number of products where a set of features appears is as easy as selecting all the features in the set in a configuration and counting its products. The result must be compared to the total amount of products defined by the SFM. So for the feature `Video On Demand` in Figure §9.2, the commonality factor is 0,724 since it appears in 504 products and the total number of products in the SFM is 696. The following code shows an implementation of this operation:

```
double commonalityFactor(SFM inputSfm, Set<Feature> featureSet) {
        SFM resetSfm = reset(inputSfm);
        int numberOfProducts = count(resetSfm);
        Set<Configuration> configSet = new HashSet<Configuration>();
        for (f: featureSet) {
                configSet.add( new ChosenStateConfiguration
                                        (f, new SelectedState ()) );
        }
        SFM configSfm = setAutoDecisions(resetSfm, configSet);
        int numberOfFilteredProducts = count(configSfm);
        return numberOfFilteredProducts / numberOfProducts;
}
```

9.4 COMPOUND EXPLANATORY OPERATIONS

The catalogue of explanatory operations for FMs in [85] links each explanatory operation to a query operation. An explanatory operation explains the results obtained from a query operation. So for example dead feature detection is linked to an operation "why is a feature dead?". In order to detect a dead feature, the SFM state is reset and a configuration that selects the feature under analysis is set. In case a feature is dead, the same SFM can be used to obtain explanations by means of a 'why are the relationships not valid?' operation. So the connection between query and explanatory operations is established by means of the configuration used to detect and explain certain behaviours.

There is a set of compound query operations that use predefined configurations or *scenarios* to detect certain behaviours. If the resulting SFM is invalid then an explanatory operation can be used to explain the source of invalidness. Query operations such as anomalies detection, void SFM, core feature and variability degree fit into the following schema:

```
1. SFM is reset.
2. A scenario is set as a configuration.
3. The list of products is obtained.
4. The output is calculated and returned.
```

If the list of products obtained in step 3 is empty then the SFM is invalid. In that case, an explanatory operation can be realised to obtain explanations why this behaviour happens. The general schema for the compound explanatory operation results as follows:

```
1. SFM is reset.
2. A scenario is set as a configuration.
3. Why are the relationships not valid?
```

Analysing the list of compound query operations in Section §9.3, six different scenarios are used in their definitions:

- **Select Feature**: It is used by dead feature detection and variability degree operations. Both operations set a configuration $\{choose(F, sel)\}$ that only selects the feature F under analysis.

- **Remove Feature**: It is used by the core feature operation. This operation sets a configuration $\{choose(F, rem)\}$ that removes the feature F under analysis.
- **Select Parent, Remove Child**: It is used by the false-optional feature detection. This operation sets a configuration $\{choose(F_P, sel), choose(F_C, rem)\}$ that selects the parent feature P and removes the child feature C.
- **Select Cardinal**: It is used by the wrong cardinal detection. This operation sets a configuration $\{choose(C_i, n)\}$ that chooses a cardinal n for a given cardinality.
- **Remove Cardinal**: It is used by the unique cardinal detection. This operation sets a configuration $\{discard(C_i, n)\}$ that discards a cardinal n for a given cardinality.
- **Empty**: It is used by the void SFM and variability operations. This scenario is a particular case where no configuration is set. In this situation, the configuration cannot be empty and perform the 'why are the relationships not valid?' operation since the resulting AP has an empty configuration that violates the precondition of the minimal explanation problem. In this case, the configuration can be set as a tautology ($\top$). Since the explanatory operation solves the negation of the observation, the configuration to explain results as a contradiction ($\bot$). So the

$$
\begin{aligned}
&WhyNotRels(SFMP(C, V, D, R, \top, S)) \equiv \\
&\equiv MinExpl(AP_C(C, V, (D, R, \bot), S, min_S)) \\
&\equiv min(\{\Delta | \Delta \subseteq R, D \not\models \bot\}, D \cup \Delta \models \bot\} \\
&\equiv MinConflicts(AP_C(C, V, (D, R, \bot), S, min_S))
\end{aligned}
$$

  With this approach we avoid the definition of a new basic explanatory operation that uses minimal conflict operation for the AP_R problem. This way the set of basic operations is kept to a minimum.

As an example, defining a compound explanatory operation to explain dead features results as follows:

```
Set<Set<Relationship>> explainDead(SFM inputSfm,GenericFeature f) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        configSet.add( new ChosenElementConstraint
                                    (f,new SelectedState()) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm,configSet);
        return whyNotRels(configuredSfm);
}
```

Compund Explanatory Operations		
Scenario	*Query*	*Why...?*
Empty	void SFM, variability	...a SFM is invalid
Select Feature	Commonality, Dead	...is F a selectable feature
Remove Feature	Core	...is F a removable feature
Select Parent, Remove Child	False-optional	...is F a false-optional feature
Select Cardinal	Wrong cardinal	...is n a wrong cardinal for C_i
Remove Cardinal	Unique cardinal	...is n a unique cardinal for C_i

Table 9.2: Scenarios and their relationship with query and explanatory operations

As a conclusion, compound explanatory operations combine model edition and basic explanatory operations to obtain explanations for unexpected behaviours in SFMs. Each scenario permits the explanation of different behaviours. The relationships among scenarios, query and compound explanatory operations are summarised in Table §9.2.

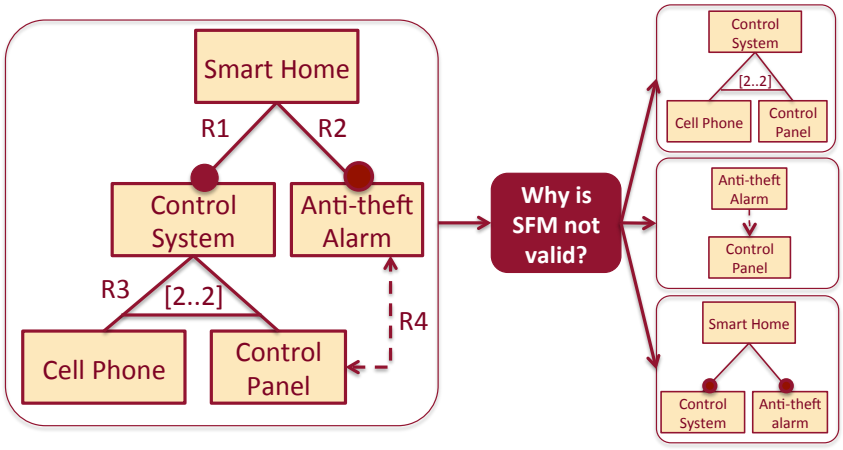


Figure 9.7: An example of compound explanatory operation for a void SFM

9.5 COMPOUND MIXED OPERATIONS

A compound mixed operation is the one that mixes query, explanatory and/or model edition operations. This category is the miscellany of operations for the AASFM. As an example, we propose an operation that detects and explains a dead feature. In case the feature under analysis is dead, explanations are provided; otherwise an empty

set of explanations is given. The definition of this operation results as follows:

```
Set<Set<Relationship>> detectAndExplainDead(SFM inputSfm,GenericFeature f) {
        Set<Configuration> configSet = new HashSet<Configuration>();
        Set<Set<Relationship>> explanations = new HashSet<Set<Relationship>>();
        configSet.add( new ChosenElementConstraint
                                    (f,new SelectedState()) );
        SFM resetSfm = reset(inputSfm);
        SFM configuredSfm = setAutoDecisions(resetSfm,configSet);
        if (!valid(configuredSfm)) {
                explanations = whyNotRels(configuredSfm);
        }
        return explanations;
}
```

9.6 SUMMARY

The proposal of a set of basic and model edition operations has allowed the definition of a wide and extensible catalogue of operations relying on their composition capabilities. We have proposed a reinterpretation of the most used AAFM operations as AASFM operations as an example of the extensibility of our proposal.

There exists an important parallelism between AAFM and AASFM. Most of the results obtained in this dissertation are applicable to the AAFM. We have obtained important conclusions for the explanatory analysis. Thus, four out of the eleven explanatory operations in the catalogue in [85], the 'why? questions, have been demonstrated to make no sense from the analysis point of view. Besides remaining operations have been defined as compound explanatory operations that use scenarios to perform the analysis. It is the scenarios that establish a connection between query and explanatory operations that was merely intuited in the original catalogue but now it is clearly stated.

The definition of a basic set of analysis operations will allow the implementation of analysis tools for the AASFM that simply propose an implementation for these subset of operations. It simplifies considerably the development effort of AASFM tools. In the next Chapter we propose a proof of concept to verify the ideas presented in this dissertation and to demonstrate the feasibility of our aproach.

Part III

Verification of Results

10

Reference Implementation

I do not think there is any thrill that can go through the human heart like that felt by the inventor as he sees some creation of the brain unfolding to success... such emotions make a man forget food, sleep, friends, love, everything.

Nikola Tesla (1856–1943),
physicist, mathematician, inventor, and electrical engineer

The objective of this Chapter is the verification showing some of the results presented in this dissertation. For that sake, we present in Section §10.1 the concept of industry-ready products and how the verification process play a key role in their construction. Section §10.2 presents an application of these concepts to build FAMA FW, a tool for the AAFM whose analysis capabilities and verification processes have inspired the construction of STEAm. STEAm is a prototype for the AASFM that is described in Section §10.3. It relies on a MDE architecture to implement the results in this dissertation. The verification process that has been applied to STEAm is described in Section §10.4. It has helped to detect errors in our results and in the implementation itself, increasing the confidence in an error-free proposal. Last, a discussion of the conclusions that we have obtained through the construction of FAMA FW and STEAm is shown in Section §10.5.

10.1 VERIFICATION, VALIDATION AND INDUSTRY-READY TOOLS

Bulilding software tools helps on verifying the research results and serves as a means of validation. With tools we demonstrate the technical feasibility of paper results. *Verification* is a process which objective is the detection of inconsistencies between the implementation and the specification. Those inconsistencies can arise due to implementation errors or errors in the specifications. Thus, verification contributes to repairing errors to produce error-free software.

When a tool is verified, it is also important to analyse its complexity and performance. A performance analysis checks that an implementation complexity fits into the theoretical complexity. It helps to determine how a tool scales for different scenarios.

With verification and performance analysis, our tools are prepared to be validated in real-world scenarios. At that point, the intervention of third-parties such as research institutions and the industry is necessary. Prototypes are commonly used to take research results to the industry. However, they are far from what the industry expects and more mature software tools are needed to shorten the distance between research and exploitation worlds. The cost and effort of developing tools is very large and they often cannot be assumed by the academy. Although verification and performance analysis improve the perception of a quality product, there still exist a gap that must be covered. So we need to take prototype to a further step that we call *industry-ready tools*. An industry-ready tool must have three desirable characteristics:

- Adaptability: industry-ready tools must target as many third-parties as possible so they must be as customisable as possible to fit into the different business needs.
- Maintainability: the industry presents a constant source of research problems. Academia provides solutions to these problems which are incorporated into tools. The academia must have the human and technical resources to keep this wheel spinning, which must in turn be supported by the industry. Industry-ready tools must allow the fastest and cheapest dissemination of research results.
- Reliability: prototypes may contain errors. To incorporate a prototype into an industrial product, it must be verified. We define testing plans to increase the trust in tools reliability. Moreover, a tool is verified by the use itself. If they detect an error, flexible architectures permit to quickly repair and deliver tool updates.

Industry-ready products are better prepared to reach the industry than a prototype, since the quality of the software is better and nearer to the quality of a commercial product. Once the gap is saved, the intervention of the industry in our validation process contributes to measure the complexity that our tool would engage with real-world scenarios. Software tools are our main means of dissemination to the industry, which helps the industry on understanding our research and helps us on capturing fundings and obtaining feedback to focus future research.

But how are we able to satisfy the three main characteristics of an industry-ready product? Architectures define the adaptability and maintainability of tools. We invest an important effort in defining flexible and customisable architectures to support both requirements. On the other hand, verification improves the reliability of an industry-ready tool.

During the development of this dissertation, we have built an industry-ready tool that has supported our research in SPLs along the last 6 years: FAMA Framework. In Section §10.2 we present how its architecture provides for the maintainability and adaptability required for industry-ready products. The definition of the AASFM as a new paradigm in the SPL world, forces us to revise all the infrastructure that is defined for FAMA Framework. We have found some limitations in FAMA Framework so that the appearance of the AASFM affords us an opportunity to define a new tool whose architecture supports both the AASFM and the AAFM. To date, we have built a prototype that we call STEAm that is described in detail in Section §10.3. It is has been built using a transformational approach very similar to the rationale followed in this dissertation. We have followed a verification process to increase the confidence in the reliability of both tools, which is described in Section §10.4. The results obtained with STEAm arises some conclusions and opens new paths to explore in the future that are discussed in Section §10.5.

10.2 FAMA Framework

FeAture Model Analyser Framework (FAMA FW) [90] is a flexible Java tool for the AAFM. FAMA FW is an open-source product and is publicly available at `www.isa.us.es/fama` since 2007 under LGPL v3 License [49]. FAMA FW transforms a FM into a suitable logical representation or *reasoner* that is used to perform the AAFM. FAMA FW is a two-sided tool. For FM analysers it is a Java library to perform the AAFM; for researchers it is a framework where research results can be easily tested and delivered

to users. FAMA FW is customisable because the user can configure it with different reasoners, analysis operations and feature metamodels. Its component-based architecture (Figure §10.1) supports different kinds of components:

- Metamodels: each metamodel component describes a feature metamodel and a general mechanism to translate it into a reasoner. EWMT Metamodel[11], Moskitt Metamodel[21] and Debian[42] metamodels are supported. Different file formats are available to store and retrieve FMs.
- Questions: a question is an interface that corresponds an analysis operations. A reasoner can implement these interfaces to support the execution of the operation.
- Reasoners: a reasoner uses an off-the-shelf solver to implement the AAFM operations. It maps any FM into a solver. Such solver may implement one or more of the available questions. JaCoP[82], Choco [66], SAT4j [15] and JavaBDD [97] reasoners are available up to date.
- Criteria Selectors: as there may be several solvers able to give a response to an analysis question, selecting the one that performs the best is important for the user to perceive an added-value service. Since knowing the solvers performing the best is not trivial, different criteria or heuristics may be taken into account. A criteria selector chooses the reasoner that is supposed to perform the best for a particular question and FM.
- Core: the common part among different FAMA FW configurations. It allows the component-based architecture that mainly communicates metamodels and reasoners, registers the available questions and use criteria selectors to search for the reasoners that may answer a question a user demands. All of this is performed keeping the independence from specific components.

Due to the orthogonal nature of the above components, a customised FAMA FW product may be configured by selecting a valid subset of the available metamodels, questions, reasoners and criteria selectors. This is the reason why we consider FAMA FW to be a SPL in itself.

From the user point of view, any FAMA FW product is a library that offers a question-answer interaction. A FM is loaded in any file format and it is asked to answer a question. An answer is provided while the user ignores which solver or metamodel

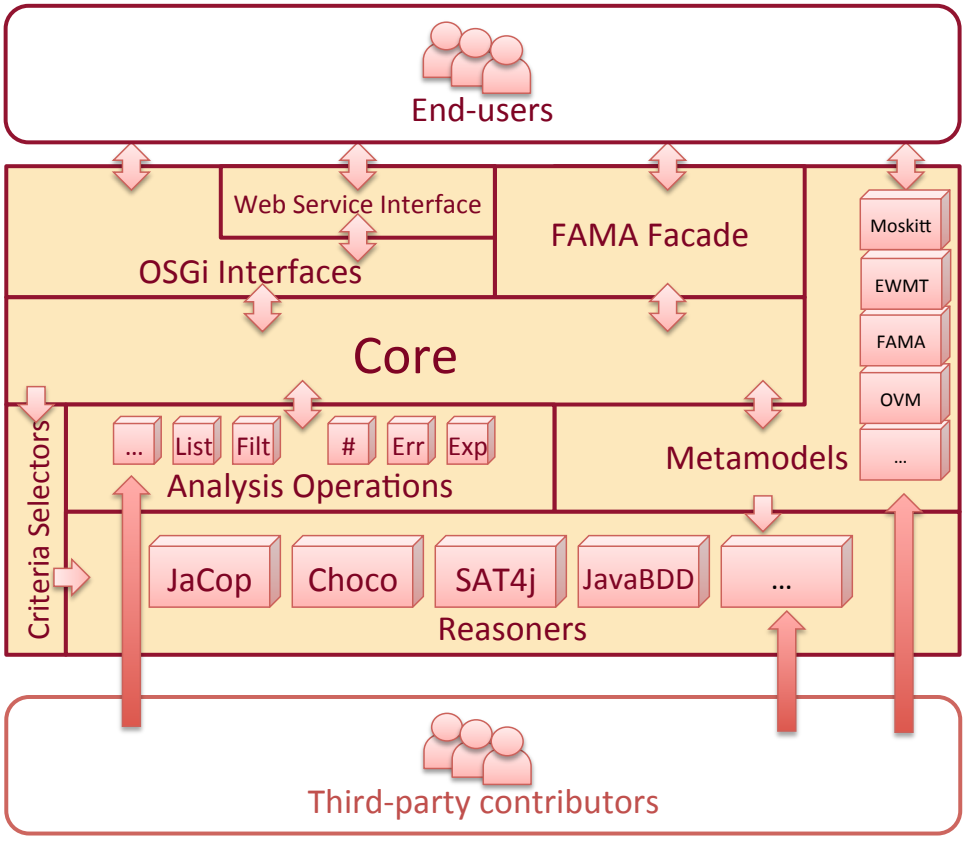


Figure 10.1: FAMA FW Architecture

has to be used to obtain it. From the point of view of a researcher, FAMA FW is a framework that offers main services to evaluate results in solvers, metamodels and criteria selectors, offering a fast dissemination of results to FM analysers.

FAMA FW has no proprietary visual editor and all its functionality is offered through a Java facade. It is prepared to be integrated into third-party tools as a Java library or as an OSGi bundle [1]. OSGI-compliance allows to consume FAMA FW from Eclipse plug-ins. We leant on this feature to integrate FAMA FW with Moskitt Feature Modeler [17] and pure::variants[72].

10.3 STATEFUL FEATURE MODEL ANALYSER (STEAM)

FAMA FW has been a successful tool that has allowed us to have a strong influence in the AAFM community. Its architecture has enabled the inclusion of research results during 6 years, however it presents some internal limitations that hinders to

accomplish the AASFM. In this Section we introduce STEAm, a tool prototype that implements the AASFM that saves FAMA FW limitations. It is important to remark that we build a prototype rather than an industry-ready tool. A prototype is a low-cost proof-of-concept that allows us to evaluate the costs and convenience of building an industry-ready tool. We expect that STEAm will soon become an industry-ready tool as successful as FAMA FW.

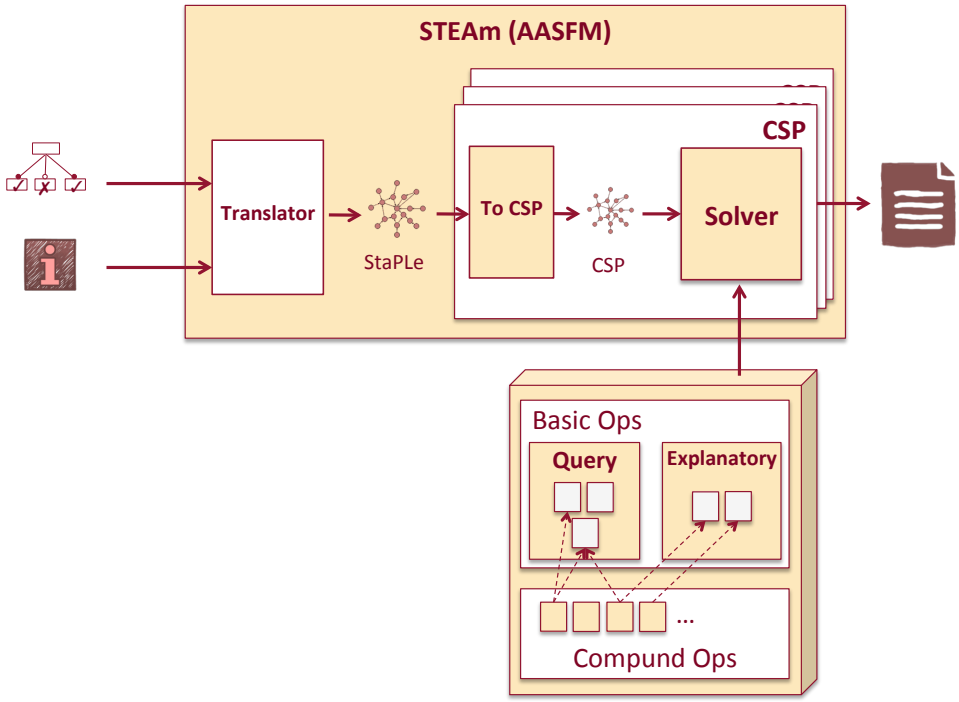


Figure 10.2: Overview on the STEAm operation

STEAm has been built using a MDE approach. With MDE, models are transformed into other models in a chain of transformations until a suitable representation is obtained. It is a natural way to implement the ideas in this dissertation and the AASFM in general. Specifically, a SFM is firstly transformed into a logical model that we call *Stateful Predicate Logic (StaPLe)*; and then into a CSP model. Figure §10.2 provides an overview on the STEAm operation. In order to reuse such transformations, they are defined in a metamodel level. So transformations can be applied for any metamodel instance, i.e. for any model. So a metamodel is defined for SFMs, StaPLe and CSP and so the transformations between them.

StaPLe is inspired in FOL and is extended and adapted with as many artefacts as necessary. The use of StaPLe as an intermediate model instead of mapping directly a SFM into CSP, introduces an indirection level that allows to define new transforma-

tions into other paradigms independently from SFMs. Thus, if another metamodel is transformed into StaPLe, the StaPLe to CSP transformation can be reused. If the StaPLe metamodel were not used as intermediate model, each input model should be transformed into CSP or any other paradigm in the prototype, increasing the implementation efforts and increasing the maintainability costs. Adding new paradigms is as easy as adding a new metamodel for that paradigm and a transformation from StaPLe. This transformation can be reused for any input metamodel that maps into StaPLe.

The so-obtained CSP model is lastly transformed into XCSP [24], a *de facto* standard to represent a CSP using XML. We choose XCSP because many CSP solvers support this format so they can be easily used and their performance checked with few effort. Since we are prototyping rather than building a tool, this design decision reduces the effort of solver integration and permits a faster evaluation of our approach. Figure §10.3 graphically describes our approach. This approach increases the maintainability of transformations since they are cohesive and first-level design entities and are not disperse in the code as in FAMA FW.

We think that our approach can be reused for other contexts rather than the AASFM. To date our research group has built ADA, a SLA analysis tool and several model analysis prototypes for BPMN models for example. To build ADA, the architecture and many artefacts in FAMA FW has been reused since it also transforms a SLA into a CSP. Although the SFM to StaPLe transformation is specific for the AASFM, the StaPLe to CSP transformation and the XCSP generation can be reused for other kinds of models. Defining a transformation from SLA into StaPLe, most of the analysis functionalities can be automatically provided by the ecosystem. We envision that this approach will contribute to reduce the maintenance and building costs of analysis tools in general.

10.3.1 Architecture

STEAm follows an MDE approach to build a chain of transformations. Its main elements in the architecture are metamodels and transformations. The input of the tool is a text file describing a SFM. It is successively transformed until an CSP in XML format is finally obtained as output. The output file is used as an input for different CSP solvers that are used to solve an analysis operation.

The architecture of STEAm demonstrates that it is possible to build a tool whose structure inspires in the rationale followed in this dissertation. In the prototype, an

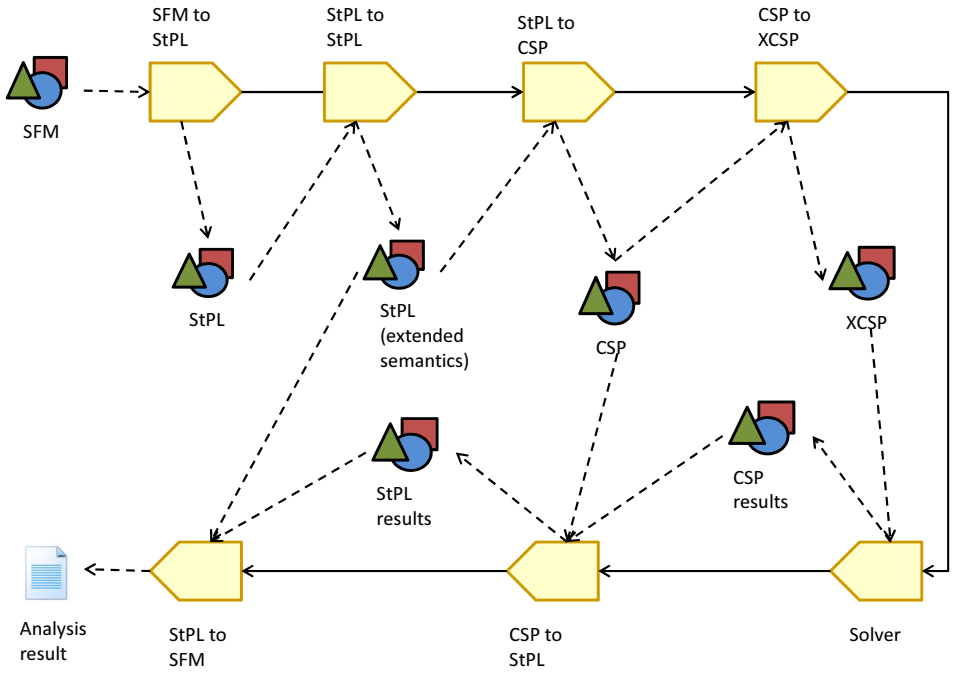


Figure 10.3: Process followed by STEAm to analyse SFMs in SPEM format

input file describing a SFM is loaded as an instance of a SFM metamodel which is transformed into a StaPLe model and then into a CSP metamodel which is converted into a XML file. Generating an XML file instead of integrating a solver has saved time and resources. The result is the same, although the automation degree is reduced. However it is enough for a prototype that is built for evaluation purposes.

FAMA FW plain-text format is the **input** format. Listing 10.1 shows an example that describes the smart home SFM in Figure §6.2. We choose this format because BeTTy [77], the tool we use for verification, uses this format.

In MDE, a **metamodel** must be defined for each kind of model that is used in the transformation process. Specifically we define a metamodel for SFM, StaPLe and CSP. As a metamodel for SFMs we use the SFMM defined in Chapter §5. For StaPLe and CSP we define specific metamodels that are described in Sections §10.3.3 and §10.3.4 respectively. Since we use the Eclipse Modelling Framework (EMF) to support MDE, each metamodel is defined as an extension of ECore metamodel (Figure §10.4).

The **output** is a CSP described in XCSP format, an XML format to represent a CSP that is widely used by CSP solvers. It permits testing the CSPs using different solvers. It allows to compare their performance and discard possible implementation errors in the solvers themselves. This way, the integration effort of the reasoners into the

prototype is avoided while our goals are still satisfied.

The **transformations** that are defined to obtain the output from the input are the following:

1. Text to model transformations: from FAMA FW plain-text format into SFMM. This transformation allows reusing FMs as inputs instead of SFMs. It facilitates the use of BeTTy for the verification process. FAMA FW parsers have been reused to create SFMs. It has contributed for a fast prototyping.

2. Model to model transformations: they are supported by *Atlas Transformation Language (ATL)*, a wide-spread technology to implement model–to–model transformations for Eclipse platform.

3. Model to text transformations: The CSP models are transformed to XCSP by means of Acceleo [69]. Acceleo is a CASE tool that permits generating code from models. We have developed a specific and reusable transformation module from CSP metamodel to XCSP format.

One of the advantages of the architecture of the STEAm is its **extensibility**. Other paradigms can be added such as SAT, BDD or description logic as a transformation from StaPLe to a metamodel for that paradigm. In case StaPLe cannot be used as an intermediate model, an *ad-hoc* transformation can be defined from SFMM to that paradigm.

Next Sections explore the details of the metamodels and transformations that have been used to build STEAm. Finally a detailed report on the verification process is presented in Section §10.4.2.

```
%Relationships
Home: Devices [Internet] [Contents];
Devices: [1,2] {HDTV42in HDReadyTV21in MobilePhone};
Internet: [1,3] {Cable Mobile3G WiFi};
Contents: [1,3] {VideoOnDemand HDD DLNA};
VideoOnDemand: [Cache] Providers;
Providers: [1,2] ProviderA ProviderB;

%Constraints
VideoOnDemand REQUIRES Internet;
Cache REQUIRES HDD;
```

Listing 10.1: A plain-text FM describing FAMA FW

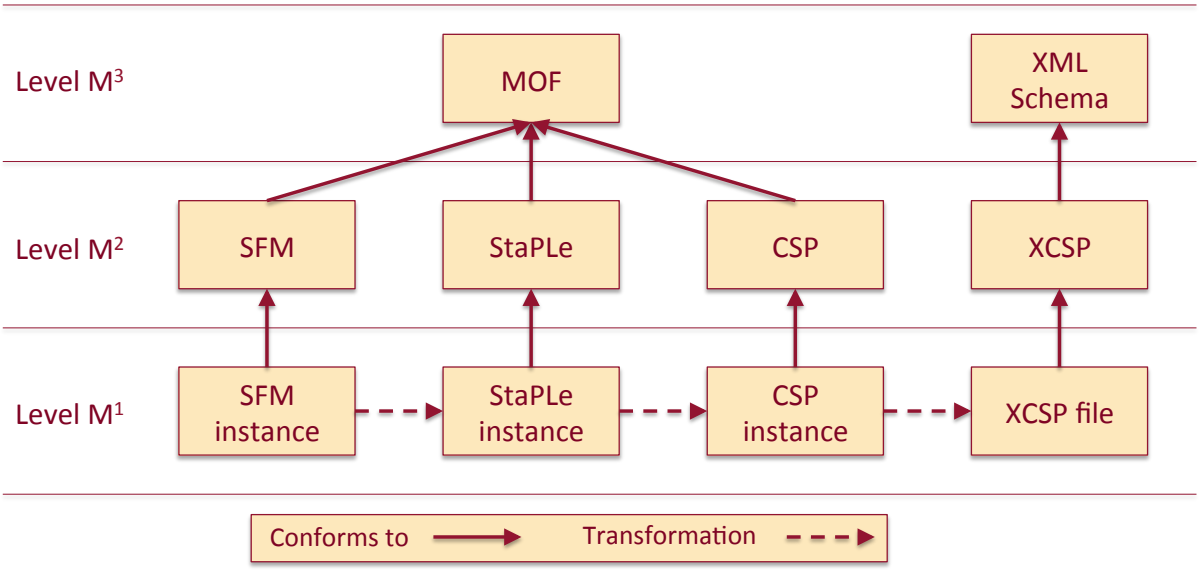


Figure 10.4: Metamodels used in STEAm

10.3.2 Stateful feature metamodel

The metamodel used in our prototype to represent FMs is the SFMM described in Section §5.5. A metamodel in EMF has to extend ECore metamodel so it can be used by the framework. It is a transparent task in EMF since the model editor automatically considers that each class in the SFMM extends `EObject`, the main class in the ECore metamodel.

10.3.3 Stateful Predicate Logic (StaPLe) metamodel

StaPLe is a metamodel to represent DPs and APs. The metamodel used for STEAm is shown in Figure §10.5. Its structure is inspired in FOL (see Annex §C), adding some classes that are needed to increase the expressiveness of FOL and to ease the transformations.

The main class in StaPLe metamodel is `Model` which is a repository of logical sentences that is divided into facts, observations and hypotheses. It permits to represent the KB for APs. DPs are described using only the set of facts, while observations and hypotheses sets remain empty.

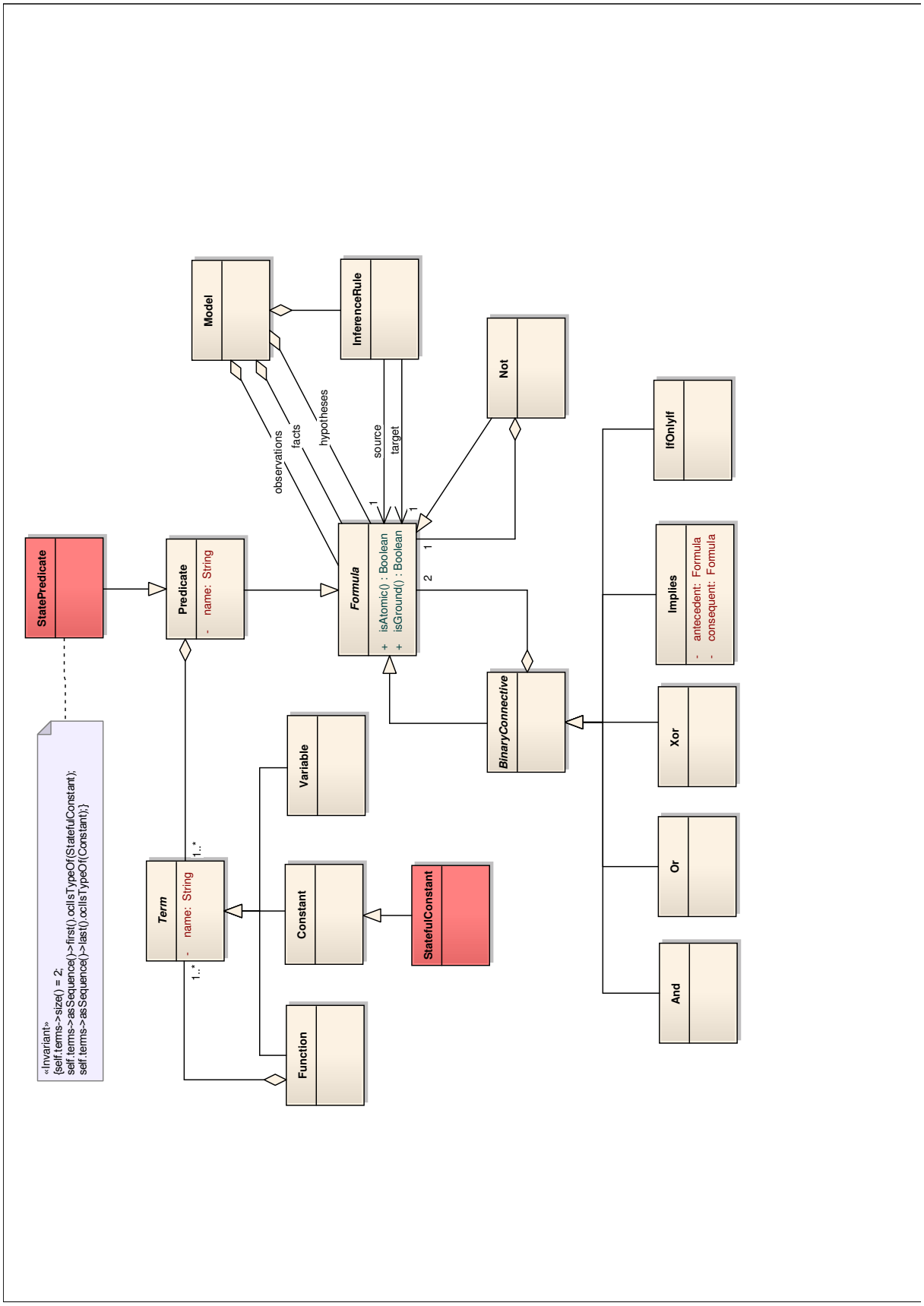


Figure 10.5: StaPLe Metamodel

Logical sentences are represented by `Formula` instances. They can be either a predicate (`Predicate` subclass), a term (`Term` subclass) or a composition of other formulas by means of binary or unary connectives (`Binary` and `Unary` subclasses respectively). Standard connectives are considered in our metamodel as `Xor`, `IfOnlyIf`, `Implies`, `Or`, `And` and `Not` classes. Universal quantifiers can also be used as `Quantifier` instances, specifically `Exists` and `Forall` subclasses that represent the existential and universal quantifiers respectively.

A predicate $p(t_1,...,t_n)$ is represented by a `Predicate` instance which stores its name and an ordered set of terms that are the parameters of the predicate. A term corresponds to a `Term` instance which can be either a function (`Function` class), a variable (`Variable` class) or a constant (`Constant` class). A function $f(t_1,...,t_m)$ also has terms so the `Function` class contains an ordered set of them. The main difference between a predicate and a function is that a function returns a result as a `Constant` instance.

A `Formula` instance is a well-formed formula. If a formula is either atomic or a ground formula, it can be determined by the methods described in this class. If the formula contains no connectives it is atomic. If no variables are affected by the formula it is ground.

The semantics is defined as a set of correspondences (`InferenceRule` class) between two formulas (`source` and `target`). The variables in each formula have to coincide in order to be correctly substituted by the terms for each appearance in the knowledge base.

We add two specific classes that are valuable for transformation purposes:

- `DecisionConstant` class is used to distinguish between constants that represent elements and states. It is necessary for further transformations.
- `StatePredicate` class is used to distinguish between *state* predicates and remaining predicates. It is used to apply the semantics and for the mapping to CSP where *state* predicates are transformed into equalities in a CSP.

10.3.4 CSP metamodel

Figure §10.6 shows the metamodel we have defined for CSPs. A CSP describes a problem (`Problem` class) in terms of a set of variables (`Variable` class) and a set of constraints over them (`Constraint` class). Each variable has a domain (`Domain` class) where it takes values from.

Constraints can be either relational (`RelationalConstraint` class) or logical (`LogicalConstraint` class). A relational constraint establishes a relationship among elements in one or more domains. Six kinds of relational constraints are defined: $\leq$ (`LesserEq` class), $<$ (`Lesser` class), $\geq$ (`GreaterEq` class), $>$ (`Greater` class), $=$ (`Equal` class) and $\neq$ (`Different` class). A logical constraint allows to represent relationships among several relational constraints. They can be either binary (`BinaryLogicalConstraint` class) or unary (`Not` class is the only unary constraint).

The expressions (`ArithmeticExpression` class) are the most complex and extensible part of the metamodel. `Variable` and `Constant` are subclasses of it so that they can be used in relational constraints like $a > b$ or $b \neq 2$. The arithmetic operators (`Operator` class) allow to represent complex relationships between variables using additions (`Sum` class), subtractions (`Minus` class), multiplications (`Mult` class) or divisions (`Div` class).

This metamodel is inspired by the XCSP schema. The XCSP expressiveness is richer than our CSP metamodel because XCSP incorporates more arithmetic and logical expressions that can be easily added to our CSP metamodel. We have reduced the expressions to those that are needed to implement the AASFM.

10.3.5 Plain-text format to SFMM transformation

This transformation is implemented as a Java bundle which parses the plain-text format and creates SFMM instances. The parser has been reused from FAMA FW and uses ANTLR technology [70].

The input format supports set-relationships, mandatory, optional, depends and excludes relationships, which are the relationships also supported by the SFMM. Although it also supports attributes they have been left out of the prototype scope.

The original input format does not support user decisions. In case they are needed, the SFM is firstly created and then the user decisions are manually set using the appropriate model edition operations.

10.3.6 SFMM to StaPLe transformation

The transformation from SFMM to StaPLe implements some of the results in this dissertation. The only remarkable consideration to take into account is that `Element` instances are mapped onto `DecisionConstant` instances and `State` instances onto `Constants`. This separation allows to distinguish between these two concepts in further

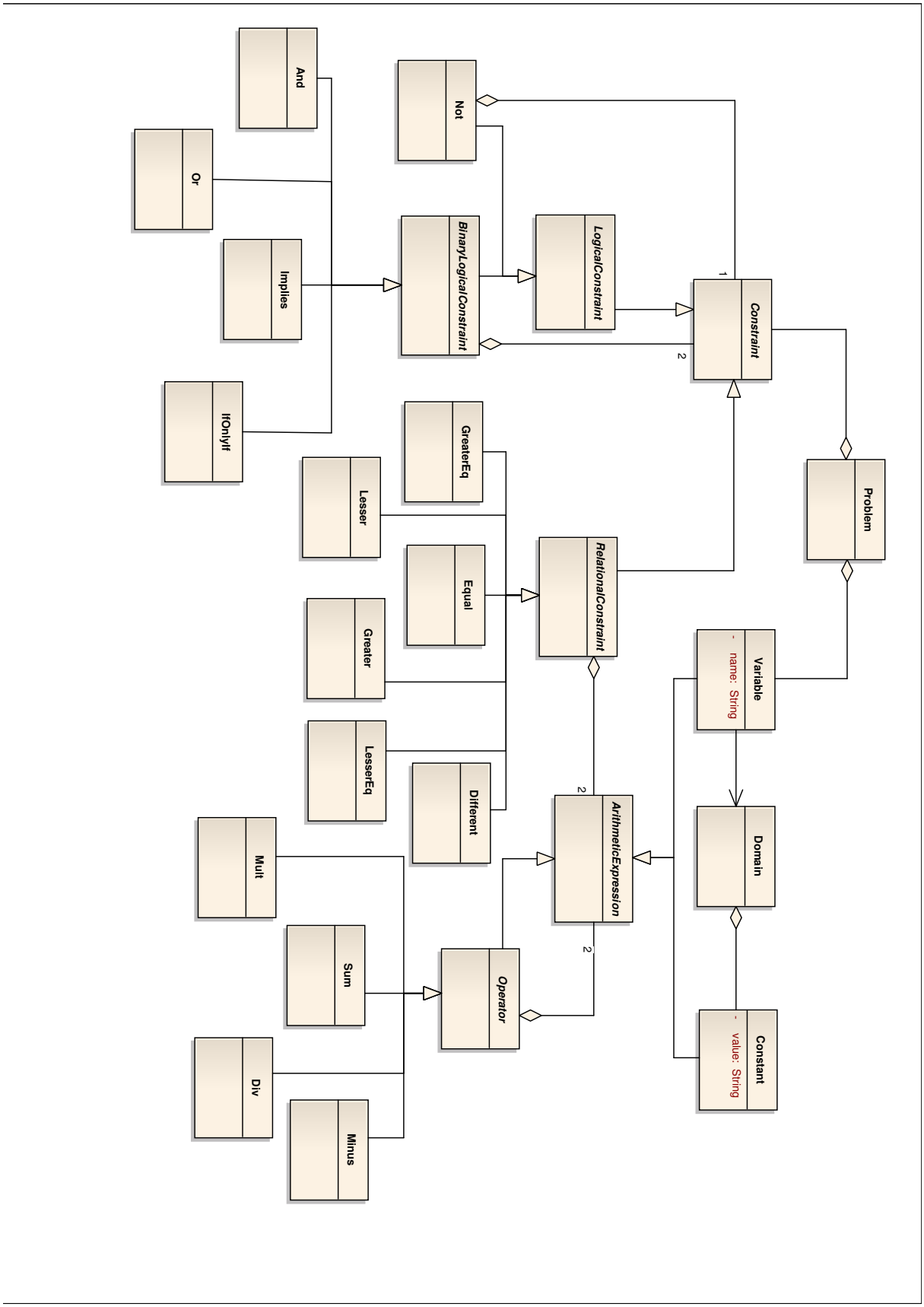


Figure 10.6: CSP Metamodel

transformations.

The semantics in the StaPLe metamodel (`InferenceRule` instances) remains the same for every input SFM and represent the correspondence between constraint and state predicates in Table §6.1.

The constant-domain mapping is also part of the semantics and is the basis of the traceability between models. There is no explicit relationship in the model between `DecisionConstant` instances and elements in the SFM. However the name of the constant can be used to search for the correspondent feature or cardinality in the input SFM.

The division of the KB in facts, hypotheses and observations depends on the kind of analysis operation to perform. So three transformations are provided to generate APs for the relationship and configuration explanatory operations and to generate a DP for query operations.

10.3.7 StaPLe to StaPLe transformation

In order to reason about predicates in the KB, non-state predicates must be replaced by its equivalence state-based sentence defined in the semantics. This transformation expands the semantics so that all the `Predicate` instances are transformed into `StatePredicate` instances.

10.3.8 StaPLe to CSP transformation

This transformation implements the results in Annex §B. The input StaPLe model must contain only `StatePredicate` instances in the KB. This prerequisite must be taken into account for reusability purposes.

There are some considerations that arise from the metamodels that intervene in the transformation which are not described in the theoretical transformation:

- `DecisionConstant` instances are mapped onto `Variable` instances while `Constant` instances in StaPLe metamodel onto `Constant` instances in CSP metamodel.
- Since the domain concept in the SFMM is disperse in the KB, the domain for a variable must be calculated from the remaining information. The domain for a variable is defined in a `Domain` instance and the values it can take are collected from all the `StatePredicate` instances in the KB of the StaPLe model such that

affect the corresponding variable.

- Since all the formulas in the input StaPLe model are composed of *state* predicates, the binary and unary connectives in StaPLe are mapped onto the corresponding logical constraints in the CSP metamodel. Each `StatePredicate` instance that appears in any formula maps onto an `Equal` instance that affects a variable and a constant.

- Formulas in the hypotheses set in StaPLe must be reified in the CSP. For this purpose, an assumption variable is created for each `Formula` instance in the hypotheses relationship in a `Model` instance. That variable is automatically assigned a boolean domain and the corresponding reified constraint is generated using that variable.

10.3.9 CSP to XCSP transformation

XCSP is an XML-based format to represent CSPs. It is widely used by all those CSP reasoners that have competed in any edition of the International Constraint Solver Competition. We have used it as an inspiration for our CSP metamodel and is thoroughly described in [24].

The transformation to text is performed by Acceleo which offers a development environment for transforming instances of metamodel classes into text. It currently supports the kinds of constraint in our CSP metamodel, although it can be easily extended to support all the kinds of constraints defined in the XCSP specification.

10.4 VERIFICATION PROCESS

Verification plays a key role to find errors in the research results and their implementation. In order to produce reliable tools we have defined a three-step process for software production:

1. Implement a solution: solutions are designed as extensions of an already existing industry-ready tool. Thus, the architecture of the tool must be as flexible as possible to incorporate future research results with the fewest modifications. It permits, when the implementation is verified, a fast dissemination of results since they are part of a deliverable product since their conception.

2. Verify the solution: the implementation undergoes a complete black-box test suite. Test suites are kept for future verifications following the principles of the test-driven development.

3. Performance analysis: the performance of the implementation is obtained and compared with other solutions. The real limits where a solution is computable are checked and compared with theoretical complexity if available.

Next we present how we have applied this process to FAMA FW and STEAm verification.

10.4.1 Verifying FAMA FW

Each step in the verification process is performed by a different tool (Figure **??**):

1. FAMA Framework (FW): its architecture permits the addition of new functionality without affecting existing elements. It is used either as benchmark product and end-user product, which accelerates the dissemination of new features.

2. BeTTy [77]: it is a framework to test any tool for the AAFM. It provides for a complete test suite for most of the existing analysis operations. BeTTy relies on metamorphic generation of sample FMs to produce a large battery of test cases. For each operation under test, BeTTy generates FMs and provides the correct results for each of them.

3. FAMA Benchmark [78]: runs and compares the performance of different AAFM tools. Although it was originally distributed as an independent product, now it is part of the BeTTy distribution.

Until the appearance of BeTTy, we had been coexisting with several errors in transformations. BeTTy has been successfully used in the validation of FAMA Framework and SPLOT[62]. In both cases, BeTTy has detected errors in the implementation of analysis operations, contributing to the debug and improvement of such tools. The combination of these tools has allowed to incorporate our advances in the AAFM into FAMA FW as extensions that have been tested and benchmarked using BeTTy.

10.4.2 Verifying STEAm

The main objective of STEAm verification is to detect problems in our proposal and implementation errors. STEAm has been built as a prototype, so the production process has been simplified compared with the process followed in FAMA FW verification. Since we have built a first version of the STEAm prototype, extensibility issues have been pushed into the background. We have also omitted the performance analysis since they are not relevant for this dissertation.

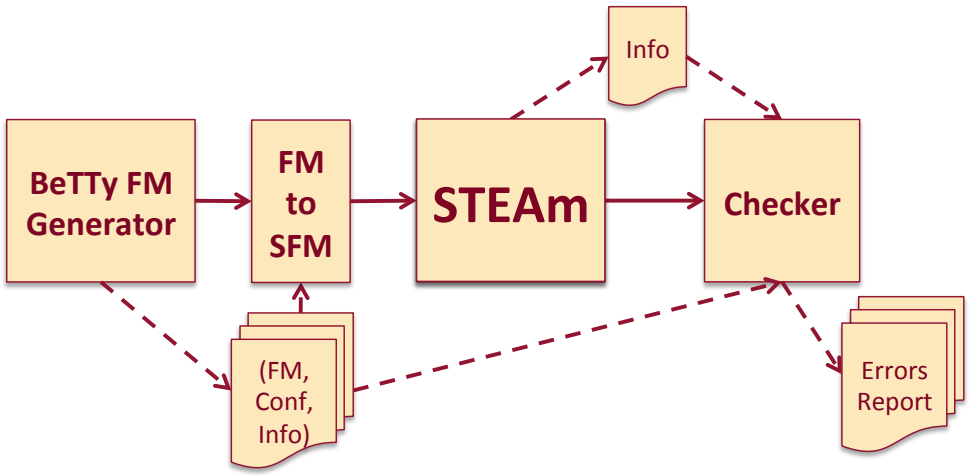


Figure 10.7: Verification of STEAm using BeTTy

BeTTy has played a key role in the verification step. It has been used to generate a set of 1000 SFMs, each of them contains 200 features, and their corresponding results for the validation operation. We have chosen validation operation since BeTTy has been conceived to work with FMs. The validation operation obtains the same boolean result for FMs and SFMs, so there is no need to adapt BeTTy for this sake. For each SFM, the following process has been applied (Figure §10.7):

1. STEAm transforms the input SFM into an XCSP file.
2. Each XCSP file is loaded by a CSP solver (Sat4j [15] in our case).
3. The solver checks if the problem is satisfiable and the result is compared with the one provided by BeTTy.
4. If a discordance is found between BeTTy and STEAm results, an error report is registered and the test case is stored for error replication.

This process has helped to find two errors, one in the implementation and another one in the specification:

- There was an error in the specification of the semantics for the *excludes* relationship. We firstly defined it as follows:

  $$excludes(f_a, f_b) \equiv state(f_a, sel) \Leftrightarrow state(f_b, rem) \wedge state(f_b, sel) \Leftrightarrow state(f_a, rem)$$

  The so-defined relationship forces to select feature `A` if feature `B` is removed, which is not the semantics of an exclusion. Both features can be removed at the same time but not selected. The following specification passes all the tests:

  $$excludes(f_a, f_b) \equiv state(f_a, sel) \Leftrightarrow \neg state(f_b, sel) \wedge state(f_b, sel) \Leftrightarrow \neg state(f_a, sel)$$

- An inefficient implementation in the transformation between StaPLe and CSP caused an out-of-memory failure. It was repaired and the affected test cases passed.

STEAm has passed the filter of BeTTy and has successfully contributed to verify and improve not only the prototype but the specification.

STEAm is the third case study where BeTTy has been applied. To date, BeTTy only supports the AAFM and query operations in particular. We plan to extend the tool to support SFMs besided FMs. Moreover, explanatory operations are not contemplated in BeTTy. Adding explanatory operations is not an easy task since the reference model used to check the results for sample FMs changes completely and obtaining it would be the result of a thorough study. We have planed to extend the tool with explanatory operations in the next future so we can also verify explanatory operations in any AASFM tool.

10.5 SUMMARY

This Chapter presents a reference implementation for the verification of the results presented in this dissertation. We firstly propose the concept of industry-ready products that guides our verification processes. When we build an industry-ready product

the destiny of every research result is being part of a software tool for results verification and dissemination. We present FAMA FW as the first tool for the AAFM which helped us to improve our dissemination and verification processes. FAMA FW has inspired the verification processes applied for the results presented in this dissertation.

We present STEAm as a prototype that supports the AASFM. It has a MDE architecture that relies on metamodels and transformations to obtain logical representations where it is possible to reason about SFMs using off-the-shelf solvers. The prototype has contributed to demonstrate the feasibility of a MDE approach and to verify the results presented in this dissertation. In the verification process, BeTTy has played a key role. It has helped to automate the product testing and has helped to detect errors in the theoretical and software results.

Building a prototype of STEAm has thrown much relevant information to build an industry-ready product. MDE is an approach that brings research and tooling nearer. Metamodels and transformations are artefacts that allow to talk in similar languages in the research and the tooling worlds, which contributes to a cheaper development and maintenance.

FAMA FW is our past and STEAm is the future of our research. FAMA FW has contributed to mature the AAFM but its maintenance is becoming more expensive as research evolves. STEAm is an alternative that incorporates SFM to the SPL world and offers better maintenance properties due to its MDE architecture. We expect that STEAm becomes an industry-ready product that allows us to disseminate our future research results and make our results visible and usable to third-parties.

Part IV

Final Considerations

11

Conclusions and Future work

A wise man shows what he's done; a fool says what he'll do.

Greek proverb,

At this point we can affirm that we have stated the basis for the AASFM as a new paradigm in the SPL arena. In this Section §11.1 we expose the conclusions we have drawn to the end of this dissertation. In Section §11.2 we discuss the pros and cons of our proposal. In Section §11.3 we propose a vision on the applicability of results on real world contexts. In Section §11.4 we envision the future work that we will accomplish at the end of this dissertation. Last, Section §11.5 summarises this Chapter.

11.1 Conclusions

The research questions that we stated in Chapter §1 aim to touch nuclear concepts in the AAFM. In this dissertation we have detected deficiencies in the AAFM regarding modelling and analysis. We dare redefine some important concepts proposed to date in the AAFM.

First, we propose SFMs as an unification of FMs and CMs to support extended configurations where users may refer to cardinalities and attributes. It is not yet another instrumental feature metamodel but a new proposal that adds new functionalities to the configuration process. Since there is a clear correspondence between SFMs and FMs, many results and tools in one side can be reused in the other side.

Second, the proposal of SFMs has enable the formalisation of all the explanatory operations proposed to date, even those that remained undiscovered. The interpretation of explanations as an AP enables the reuse of existing works in the artificial intelligence community.

Third, we have extended the formalisation efforts to non-explanatory operations, interpreting them as DPs. This approach together with the formalisation of explanations has allowed to define a subset of basic analysis operations on top of which remaining operations are defined as a composition of them. This approach enables the definition of new analysis operations that might arise in the future that rely on this composition mechanisms. Moreover, it eases the construction of analysis engines since only basic operations and composition mechanisms need to be implemented to provide a full support.

Last, we have built STEAm, a prototype tool that demonstrates the feasibility of these results. We have used an MDE approach, which is novel in the SPL community for the automated analysis.

With these results, we can affirm that in this dissertation we set the basis for the AASFM.

11.2 Discussion and limitations

Proposing a new concept such as the AASFM has a high risk of rejection by the community. The AASFM formalises more analysis operations than the AAFM, and pro-

vides the extensibility mechanisms to compose new analysis operations on demand. The success of the AASFM goes through exploiting the correspondence between SFMs and FMs, which allow to reuse the results and tools in both sides.

One of the main limitations of joining FMs and CMs in a single SFM is that they can only store one configuration at a time. Under some circumstances we could need more than one configuration. However, one of the main benefits of SFMs is their analysis capabilities. Probably, as SPL developers we should wonder if FMs or SFMs are suitable for modelling the products, supporting the configuration process and assisting the SPL development. Finding a model that fits all the problems that arise in each of these applications can be an unfeasible task. For this sake, we shall propose specific models that support each of these applications, building transformations among them. In this situation, the SFMs are our proposal for a better support of the AASFM.

Although the abstract model proposed in this dissertation for SFMs consider the incorporation of cardinalities and attributes to SFMs, we have discarded incorporating attributes to our semantics to avoid increasing the complexity of the problem to solve even more. We have built a solution for discrete attributes and a tool prototype, however the introduction of continuous attributes would introduce more complexity in our approach, so we have decided to leave them for a future work.

We are conscious that the number of transformations that we propose in this dissertation could be reduced. The adoption of SFMP for example, as a generalisation of the mapping from SFMs to DAPs could have been avoided. However we decided to keep them in the dissertation to avoid repeating mapping rules from SFMs to DPs and to the two APs. We intuit that SFMPs, DPs and APs could be joined in a unique model. As a proof of concept we have defined StaPLe in STEAm that we have used as an indirection level between SFMs and each declarative language.

11.3 Applications

The construction of a prototype tool increases the applicability of our results. Our experience in building FaMa Framework tells us that building tools increases the visibility of our results and the interest of the research community and the industry.

The experiments realised to date with STEAm do not provide good results in terms of performance. The choice of MDE as a runtime platform could not be the best one. Most of the MDE tools are not mature enough to be deployed in realtime environ-

ments. The lightweight architecture of FAMA Framework is still a better solution that STEAm for real world applications. However, we think that the MDE approach still has something to do with the AASFM. We think that MDE could be used to create the source of specific AASFM tools instead of being the tool itself.

As we explore the best choice for the next generation of analysis tool, FAMA Framework still offers many benefits that we can exploit. The addition of SFMs as input models and the revision of its internal architecture to adopt new analysis operations is a priority for us.

We have already added CMs as a separate model in FAMA Framework to support configuration explanation in ISA Packager [43], a tool for the automated deployment and maintenance of SCADA systems. It was one of the first steps towards SFMs and an inspiration to propose the unification of SFMs and CMs.

At the time of writing, we are applying SFMs to model the infrastructure of cloud computing systems [44]. It enables the use of AASFM operations to support the configuration process of the *Infrastructure as a Service* (IaaS), choosing the configuration that suits the best in user needs. For this purpose, attributes are intensively used to represent costs and measurable properties such as data transfer, hard disk consumption or data storage.

11.4 Publications and Future work

We are aware of the number of results in this dissertation that have not been previously published. The previous publication of important intermediate results such as [85, 89, 99] has given us the opportunity to use this dissertation as a revision of the bases of the AAFM. The publication of these results and their validation by the community will be our goal in the coming months.

Giving a full support to attributes is one of our main concerns. Moreover when we aim to apply our results for cloud computing systems where attributes play a key role. The publication of our approach for the AASFM extended with a full support for attributes is a priority for us. We think that the new operations that can arise from their treatment will open an exciting new field to explore.

Our contribution has been designed thinking in its future extension. We have provided extension mechanism in the SFM to accept new kinds of elements, relationships

and decision constraints. Moreover, the use of DAPs allow the development of different implementation approaches. Our research efforts will focus on exploiting the extensibility of our proposal, firstly focused on attributes.

The introduction of DAPs is a novel approach that we think that may ease the construction of analysis tool of any kind of model. From our previous experience in developing analysis tools [91] we have found connection points between the analysis of SFMs and other models such as *Service-Level Agreements (SLAs)* or *Business Process Model and Notations (BPMNs)*. We want to explore these connections in order to build a single tool supporting the automated analysis of these model.

To apply our ideas it is fundamental to have feedback from real-world users and to discover new problems to solve. Despite the performance results obtained from STEAm, we aim to invest more resources in STEAm in order to transform it into an industry-ready tool. STEAm has shed light on the feasibility of building an ecosystem to support the automated analysis of models of different kinds, not only SFMs. We think that many of the transformations and models that have been proposed here can be reused to analyse other models rather than SFMs. A model-driven architecture and the use of transformations are the approach whose benefits we want to explore.

11.5 SUMMARY

As a last summary and conclusion of this dissertation, we might affirm that we have set the basis of the AASFM, an evolution of the AAFM that solves many deficiencies in modelling and analysis. We have proposed SFMs as a new kind of model that solves modelling deficiencies in the AASFM and a formalisation framework for the AASFM, a new paradigm to analyse SFMs. Despite we have left attributes out of the scope of this dissertation we are on the extension of our approach for a full support of them. We propose a future plan to disseminate these results that consists on releasing selected papers that save the limitation regarding attributes and building an industry-ready tool for the AASFM.

Part V

Appendices

A

Stateful Feature Metamodel Design Documents

A.1 Requirements

We organise the objectives presented in Section §5.5 as a set of information (IR), functional (FR) and non-functional (NFR) requirements:

IR1. The metamodel must interpret a SPL as a set of elements that are constrained by means of relationships and user decisions.

IR2. The metamodel must represent the states for each element, either features or cardinalities.

IR3. The metamodel must support different kinds of constraints for relationships and user decisions.

IR4. The metamodel must represent the decisions each user makes on each of the elements in a model.

FR1. Defining methods for the model edition operations in Section §5.5.1.

NFR1. The kinds of elements and constraints in the metamodel might vary in the future, so the metamodel must support the addition of new kinds of elements and constraints.

NFR2. The consideration of a model as a set of constraints defined on a set of elements might fit into other models. The metamodel must be as general as possible to be reusable in other contexts in which our research group works in such as SLAs and BPMNs.

These requirements guide the design of the metamodel. So IRs generate classes and attributes; FRs guide the definition of methods; and NFRs justify the use of generalisations.

A.2 Design decisions

Figure §A.1 depicts the SFMM that has been obtained as a consequence of the design decisions described in this Section. To obtain this design, three steps have been followed: First, generalisation is taken into account. With generalisation, we envision the probable changes in requirements and the appearance of new ones. Solutions designed relying in the generalisation principle supports many changes as they appear, reducing the need of changing existing artefacts and prolonging the lifetime of the solutions. Second, the information to be stored by SFMs is defined, thinking in the addition of new elements.Third, model edition operations are added once the classes and interfaces structure is closed.

Next we enumerate the main problems in SFMM design and how they have been solved. The following decisions are presented in a format inspired by technical memos [26, 56] which are an exhaustive form to explain a solution given in design. We approach the problem, how it has been solved, why we have solved the problem that way and optionally which are the alternative solutions that have been considered and why they have discarded.

A.2.1 Stateful feature models are sets of elements and constraints

Problem

Designing a solution for IR1. Specifically, representing a SFM as a set of elements and constraints among them. The solution must be general enough to support as many future extensions as possible as remarked by NFR2.

Solution

Figure §A.2 depicts the solution given for this problem. A SFM aggregates elements and constraints. Each element can have one or more states. Constraints are used to define the valid state combinations among elements. Stable interfaces are defined to solve this problem. Specifically, the concept of a *stateful model* is defined to support future extensions in SFMs and in other models that rely on the element-constraint structure.

Four classes are proposed: `StatefulModel`, `Element`, `State` and `Constraint`. The `StatefulModel` class acts as a mere container of constraints and elements. An element has a domain, which is the set of states it can have. It is stored in the `domain` aggregate of the `Element` class. A `Constraint` class stores in the `affectedElements` aggregate

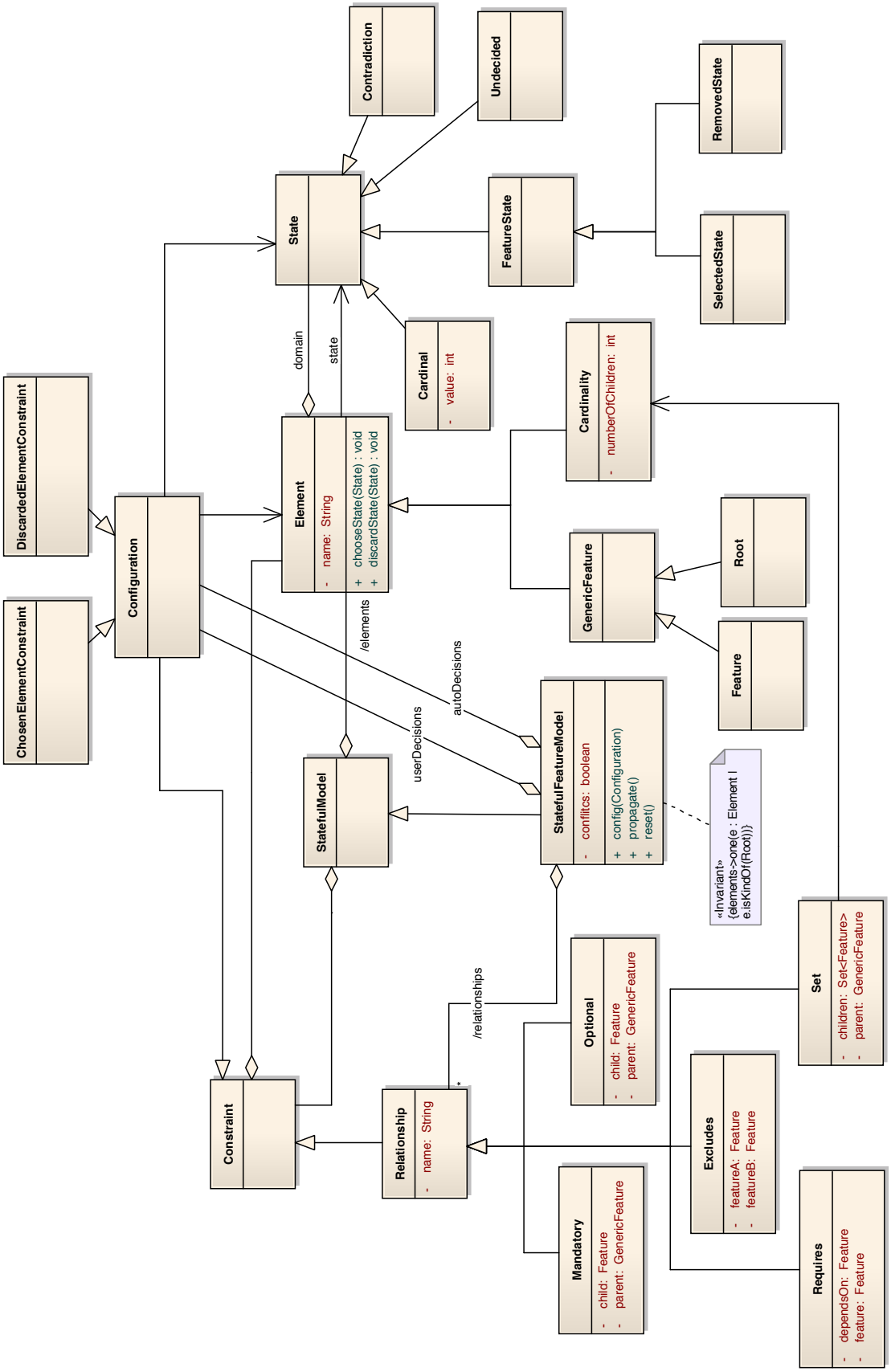


Figure A.1: Stateful Feature Metamodel in UML format

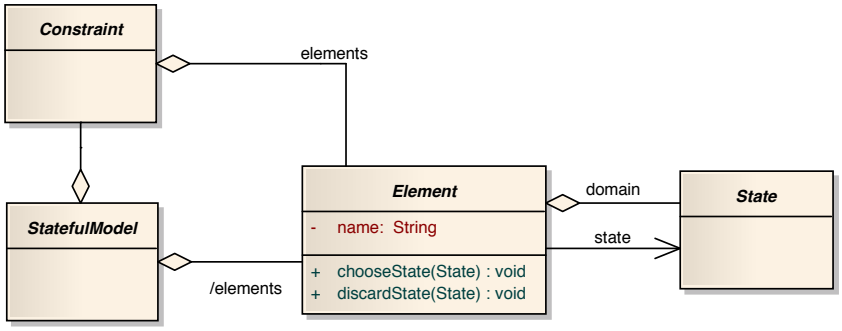


Figure A.2: Generic classes of the SFMM

attribute one or more elements that are affected by the constraint.

All the classes in the stateful model are abstract to avoid direct instantiation. Domain-specific classes must refine them by means of inheritance, adding the particularities of its domain.

Justification

To generalise the solution, the protected variations GRAS principle [56] has been applied to identify the evolution and variation points in the metamodel and to create stable interfaces. For the sake of generalisation, we define a model that focuses in the behaviour of the elements as much as in the constraints among them. Elements are the generalisation of features, cardinalities and attributes; States are the generalisation of selected, removed, undecided and contradictory states; Constraint is the global term to refer to relationships and configurations which affect different kinds of elements in different ways.

A stateful model is a generalisation that can be thought to be useless. In the last years, many different feature metamodels have been proposed. Moreover, there exist other kinds of variability models. Decoupling the element-state-constraint entities from the SFM allows to reuse this structure with other variability models that could be interpreted in terms of elements, states and constraints.

A.2.2 Representing specific elements and states

Problem

Designing a solution for IR2, taking into account FR1. Specifically, representing features and cardinalities as elements and their corresponding states.

Solution

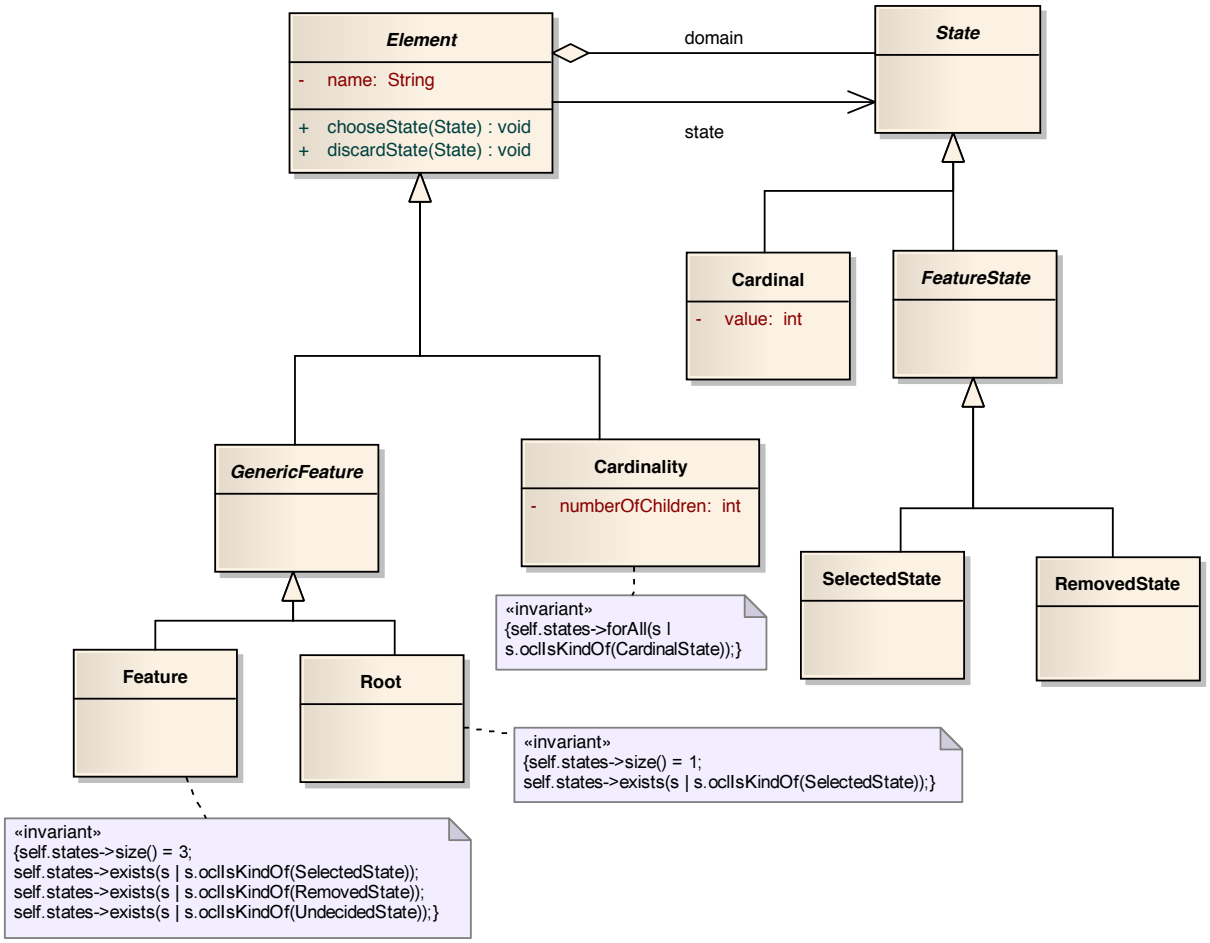


Figure A.3: Features and cardinalities in the SFMM

Figure §A.3 depicts the solution given for this problem. For each kind of element a parallel hierarchy of classes must be created: one for the kind of element itself that inherits from the `Element` class and another one for each specific state it can have that inherits from the `State` class.

For features, `Feature` and `FeatureState` classes are defined. One child class of `FeatureState` is created for each feature-specific state: `SelectedState` to represent the selected state and `RemovedState` to represent the removed state

The root feature is modelled as a separate `Root` class. A root feature must appear in any product so it can only have the selected state. This restriction is represented by an OCL constraint in the UML model. `Root` and `Feature` classes share a common `GenericFeature` superclass that is used when it is necessary to refer to any feature in a

SFM.

For cardinalities, `Cardinality` and `Cardinal` classes are defined. A cardinality contains a set of cardinals, each of which represents a state of the cardinality. Each state is a cardinal defined in the range of a cardinality.

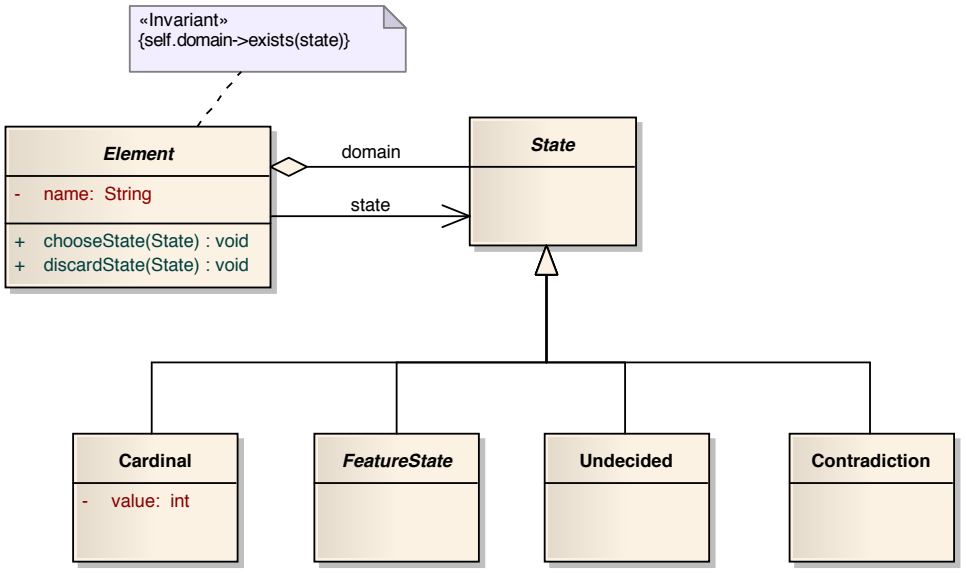


Figure A.4: Elements lifecycle

Besides the element-specific states, two states are added to represent undecided and contradictory states. They are modelled as `Undecided` and `Contradiction` classes which extend the `State` class as depicted in Figure §A.4. Every element has a current state which can be either an element-specific, undecided or contradictory state. So for example, a feature can either be in a selected, removed, undecided or contradictory state.

Two operations are defined to manipulate states: `chooseState` and `discardState`. They correspond to the actions that are allowed to perform with an element. Initially any element has an undecided state. Whenever a state is chosen, it cannot be directly assigned in every situation. If the previous state is undecided, the current state changes normally. It the previous and chosen states coincides, the state remains. In any other circumstance, the previous state will be different to the chosen state so a contradiction arises. A contradiction is pointed by a `Contradiction` instance as the new current state.

Justification

The extension mechanisms provided by the stateful model are exploited in this solution. Features and cardinalities have been modelled using the element-state duality.

`FeatureState` has been defined in order to support future addition of new feature states. It could be useful for dynamic SPLs [19, 46, 88] where active and inactive feature states can arise at runtime to indicate a feature availability.

`GenericFeature` is an abstract class that is defined to refer indistinguishably to root and non-root features. This distinction will arise benefits for the definition of relationships in Section §A.2.3.

Users interact with the model changing the states for elements. If an element has never been assigned a state, it is said to be undecided. Contradictory assignments must be allowed since there could be more than one user making decisions. So a user can select a feature that is discarded by another user. In this case, a contradictory state is marked for that feature. Since the metamodel works with contradictions, it allows multiple users configuring at the same time in the so-called parallel or staged configurations [99].

Alternative solutions

- As an alternative to `GenericFeature` class, we firstly considered to model the `Root` class as a child class of `Feature`. However it is known to be a bad smell in design [41] known as refused bequest. It considers that defining a child class that refuses part of the information and behaviour of its parent is a bad practise. Under this circumstance it is recommended to create a parallel hierarchy and create a common parent class with the shared behaviour. As a result we have preferred to define `Feature` and `Root` classes as children of the `GenericFeature` parent class.

- We considered defining feature states as enumerated values in a `FeatureState` class. However it forces to modify this class every time a new state arises, violating the open-close principle. The given solution is less invasive at the cost of increasing the number of classes in the metamodel.

- Instead of defining a domain and a current state, it was also considered to distinguish among available, discarded and current states. Initially the available set corresponds to the domain. Whenever an user chooses a state for an element, it is assigned as the current state and remaining available states are discarded. An user can also discard a state that is not valid anymore, moving the state from

the available to the discarded set. This solution was discarded since it impedes working with contradictions .

- The current state could have been modelled as a `0..1` association instead of defining an undecided state. However the use of null references is considered to be a bad practise and error-prone approach. A `Undecided` instance plays the role of a null object.

A.2.3 Representing relationships as constraints

Problem

Designing a solution for IR3, taking into account NFR1. Specifically, representing the different kinds of relationships as constraints on the set of elements, opening the door to future addition of new kinds of relationships.

Solution

Figure §A.5 depicts the solution given for this problem. The defined kinds of relationship affect features and optionally cardinalities.

A relationship is a particular kind of constraint. It is represented as an abstract class `Relationship` which is a subclass of the `Constraint` class. Each kind of relationship is supported by an extension of the `Relationship` class. Five kinds have been defined to represent mandatory, optional, set, excludes and requires relationships.

Justification

Feature metamodels in the literature use different kinds of relationship to represent products variability. We propose five different kinds of relationship which are the most commonly used in the literature and in tools. Other kinds of relationships can be added as implementations of the `Relationship` abstract class.

The benefits from the distinction between root and non-root features arises with relationships. Wherever `Feature` class is used instead of `GenericFeature`, it avoids root features to participate in relationships in a way that introduces error or unwanted behaviours in a SFM. This approach affects the relationships as follows:

- `Mandatory` and `Optional` classes: they represent two hierarchical and binary relationships. They link a parent and a child feature. Due to the hierarchical structure

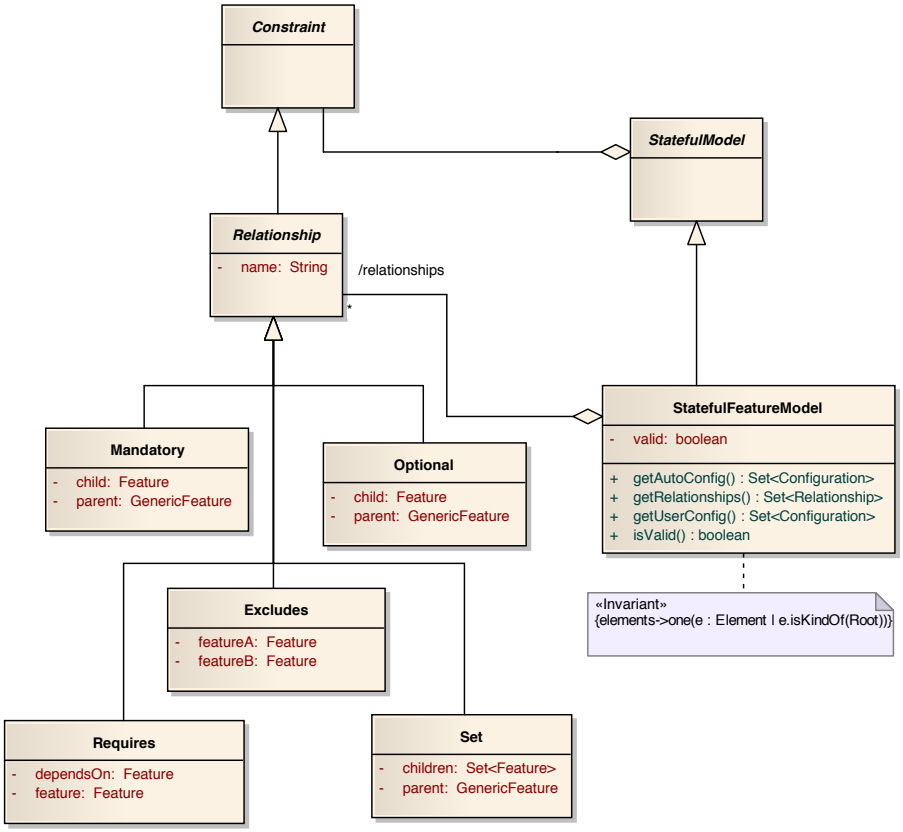


Figure A.5: Kinds of relationship

parent features are `GenericFeature` instances while child features must be a `Feature` instance. This way, a child feature is avoided to be the root.

- `Set` class: a hierarchical relationship linking a parent `Feature` instance to a set of child `Feature` intances. It is affected by a `Cardinality` instance that constraints the number of child features that can have a selected state at the same time. As for mandatory and optional relationships, child features cannot be root features so they are instances of `Feature`; meanwhile the parent feature can be the root feature so it is an instance of `GenericFeature`

- `Excludes` and `Depends` classes: root features are not allowed to participate in cross-tree constraints since they are a source of contradictions and false-optional features [89]. Therefore the two features intervening in depends and requires relationships must be instances of `Feature` class.

A.2.4 Representing user decisions as constraints

Problem

Designing a solution for IR4, taking into account NFR1. Specifically, representing user decisions as constraints. The solution must ease the addition of new kinds of decision constraints in the future.

Solution

Figure §A.6 depicts the solution given for this problem. A user decision is a kind of constraint that is represented as a `Configuration` class. Since a user decision affects one element and one state, they contain `Element` and `State` instances. Any kind of user decision is defined as an implementation of the `Configuration` class. Two kinds are proposed: `ChooseStateConstraint` and `DiscardStateConstraint` classes which are used to represent the selection and discarding of an element state.

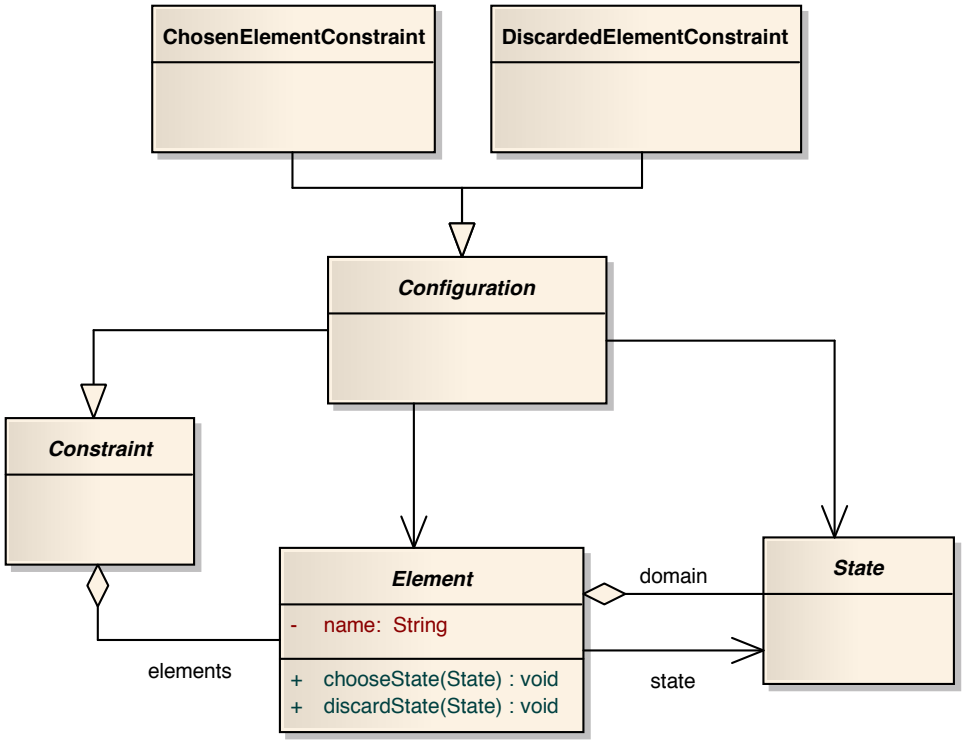


Figure A.6: Kinds of configuration constraints

Justification

The current state for an element is not enough to store all the information about user decisions. One or more users can add decision constraints that contradict or repeat themselves. Element states are not rich enough to represent this information. So

for example if two users select the same feature, its state is selected. The same state that results from a unique configuration. In case a feature is in contradiction, the contradictory decisions made by users must be registered.

So the `Configuration` class is used to register and constrain a state in an element domain. Generalising `ChooseStateConstraint` and `DiscardStateConstraint` by means of the `Configuration` class permits to accept eventual new ways of representing the selections made over the states.

Open issues

Configuration constraints can be set by a user of the model or by automatic means. An example of an automatic configuration is the propagation operation that selects automatically and based on the relationships within the model, the state of those elements which can only have a unique valid state. It is important to distinguish between user and automatic configurations in order to explain and repair contradictions in the model. In this situation, user decisions are the source of the contradiction and must be the only configurations that must be repaired. Automatic configurations can be always recalculated from the information contained within a SFM.

A.2.5 Generalising stateful feature models

Problem

Designing a solution for NFR2. Specifically, generalising SFMs to be reusable in other contexts. The solution must distinguish among user and automatic decisions.

Solution

Figure §A.7 depicts the solution given for this problem. A class `StatefulFeatureModel` refines the `StatefulModel` class. Three methods are added to obtain the set of relationships, user configurations and automatic configurations from the set of constraints stored in the parent class: `getRelationships()`, `getUserConfig()` and `getAutoConfig()`.

A boolean attribute is also stored to mark an invalid SFM whenever a contradictory state is set for any element in the model. A method `boolean isValid()` permits to obtain if any contradiction has been found in the set of constraints.

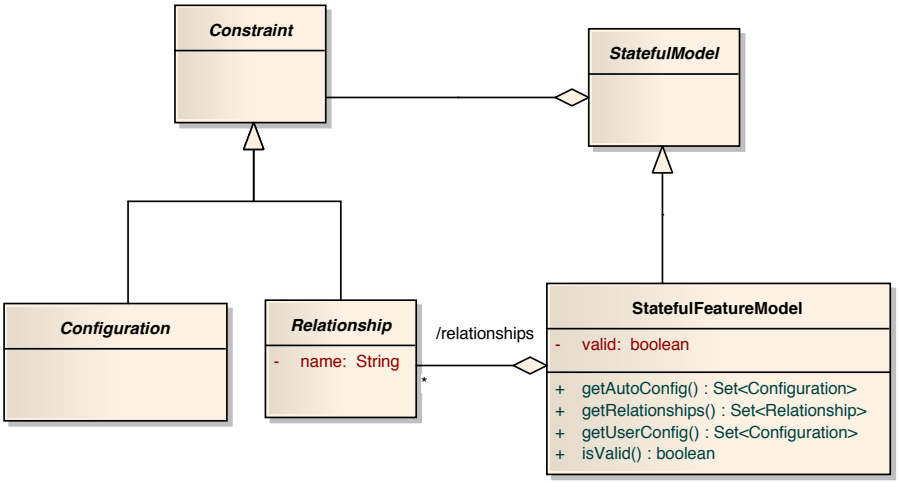


Figure A.7: Stateful Feature Model class in detail

Justification

The `StatefulModel` class stores the set of constraints in the model. In SFMs it is mandatory to distinguish among relationships, user decisions and automatic decisions. It is possible to separate the set of constraints into two sets of relationships and decisions from the classtype. However it is not possible to distinguish between user and automatic decisions from the class hierarchy. So the `StatefulFeatureModel` class stores the set of automatic decisions separately and calculate user decisions and relationships from the constraints set in the `StatefulModel` class.

The set of elements is provided by the parent class so there is no need to change or extend this relationship.

Alternative solutions

Relationships, user configurations and automatic configurations could be stored in three different aggregates, refusing the parent aggregate for constraints. It is a case of refused bequest, a bad design practise which must be avoided whenever possible.

A.2.6 Collecting user decisions

Problem

Designing a solution for FR1. Specifically, defining methods for the model edition operations.

Solution

We propose a set of operations to manipulate user and automatic decisions based on the use we are making of decisions for the automated analysis of SFMs. Two methods are defined to add sets of user and automatic decisions. The only way to remove decisions is resetting the SFM, which eliminates all the decisions and restores the state for every element to undecided.

They are explained in detail in Section §5.5.1

Justification

We could have added standard methods to manipulate data aggregates such as addition of elements, removal, search, etc. However we have preferred to reduce the set of methods to those that we are certain about their use. We do not want to create methods that are not be used.

A.3 EXTENSIBILITY

The extensibility of the metamodel is given by the variation points that have been considered in the metamodel design. Next we present the extension methods that are provided by the metamodel. An example of extension is presented whose objective is to show the extension mechanisms and to present an overview on how we pretend to extend the metamodel to support attributes in the future.

A.3.1 Variation points

The model presents four main variation points: elements, states, constraints and models. Any new issue must fit into any of these four concepts. Extension is provided by inheritance. So a new element must extend the `Element` class. In parallel, new states must be defined as extensions of the `State` class. The domain of the element must be created to contain the new states. New relationships can be added as an extension of `Relationship`. New configuration constraints as an extension of `Constraint` classes.

We think that there exist many models in other contexts that can be defined in terms of elements that have states and constraints over them. We envision that It will allow to reuse our analysis proposal in other contexts further than SFMs. Our metamodel supports them as an extension of the `StatefulModel` class.

A.3.2 An example of extensibility: supporting attributes

In order to illustrate how the SFMM can be extended, we propose adding discrete attributes to the metamodel. A discrete attribute is a feature property that takes values in a finite and discrete domain. An attribute is linked to a feature and is affected by its state. So for example, a `PrinterDriver` feature has an attribute `memory` that represents the memory consumption. It can consume either 1Mb working in slow mode or 3Mb working in fast mode. If the `PrinterDriver` feature is removed, then the consumption is null (zero state).

So there are three artefacts that must be added to the SFMM: a new element and new states for attributes and a new relationship to link the state of the attribute and the state of its feature. We define a `DiscreteAttribute` class as an extension of the `Element` class. The domain for this class contains one state for each valid value. We define a `DiscreteAttributeState` class that extends the `State` class to store those values.

The link between feature and attribute can be represented as a constraint. An `AttributeRelationship` class extends the `Relationship`. This class stores a constraint that obligues to set the the zero state if the corresponding feature has a removed state or permits to select any other value in the attribute domain in case the feature is selected. The class is associated with a `GenericFeature` and a `DiscreteAttribute` instance. Figure §A.8 depicts the new elements and ther integration into the SFMM.

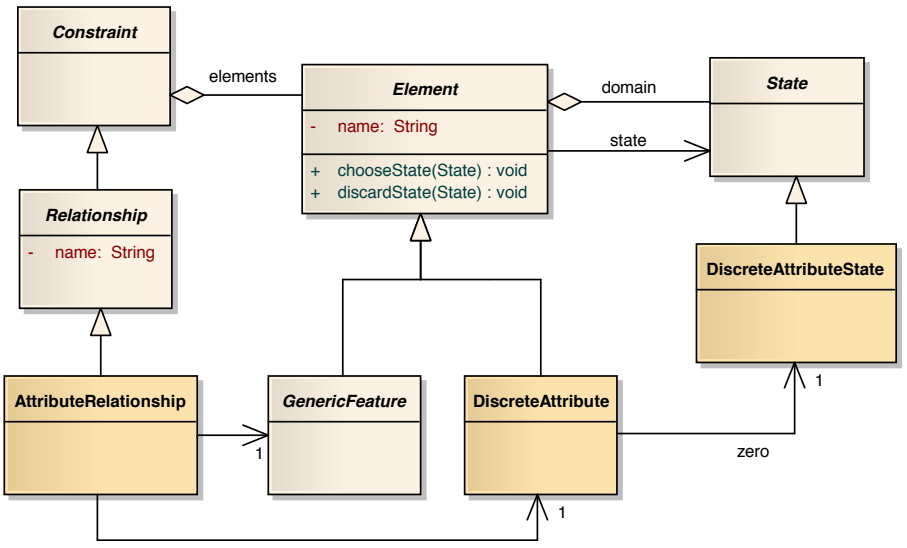


Figure A.8: An example of extension of the SFMM for a basic attributes support

This is a very basic approach. The complexity of dealing with different data types in attributes such as strings, integers, floats or enumerations is not solved with our toy example. To date there is no thorough study about the complex relationships that may arise from the use of attributes. We have introduced a basic support of attributes in FAMA FW and have defined a textual description of FMs where attributes can be defined. We will work in the future to extend the metamodel with a better support of attributes and all the functionality around them.

B

IMPLEMENTATION

A fact is a simple statement that everyone believes. It is innocent, unless found guilty. A hypothesis is a novel suggestion that no one wants to believe. It is guilty, until found effective.

Edward Teller (1908–2003),
Nuclear physicist

In this section we summarise some of the techniques that can be used to implement the AASFM, with a special focus on the explanatory analysis. Two proposals are presented in this Chapter. Firstly in Section §B.1, the FOL representation can be naturally mapped onto a constraint programming problem which is a traditional approach in the AAFM. Second in Section §B.2, default logic is proposed as a non-monotonic logic that can be used as an abductive reasoner. These two solutions have been chosen to represent two different ways of implementing abduction, using monotonic and non-monotonic reasoners. Last, Section §B.3 provides an overview on other alternatives that are used in the artificial intelligence community to support abduction and that we think that could be applicable to explanatory operations. Deductive reasoners have been widely used in the AAFM [14] so they are left out of the scope of this Chapter.

B.1 CONSTRAINT PROGRAMMING

A CSP is composed of *variables* $V = \{v_1, \cdots, v_n\}$ each of them ranging into a finite *domain* $D = \{D_1, \cdots, D_n\}$. Among those variables there exist *constraints* (C) that reduce the valid combination of domain values. A solution to a CSP is an assignment of values to variables in their domains $\{(v_1 \mapsto x_1, \cdots, v_n \mapsto x_n) | \forall i, x_i \in D_i\}$ that satisfies all the constraints in C.

For example for a CSP $((V,D),C)$ where $V = \{a,b\}$, $D = \{\{0,1,2,3\},\{0,1,2,3\}\}$ and $C = \{a > 0, a + b = 2\}$ there are two solutions $\{a \mapsto 1, b \mapsto 1\}$ and $\{a \mapsto 2, b \mapsto 0\}$ that satisfy the constraints. A CSP can have many solutions each of them as relevant

as others. A COP [33] introduces an optimisation function to a CSP so it only seeks for those solutions that maximise/minimise that function. So for a COP maximising the function $O = a$, it only obtains $\{a \mapsto 2, b \mapsto 0\}$ as a solution.

B.1.1 Mapping from FOL to CSP

The following mapping generates a CSP in two steps. Firstly variables and domains are generated from the constants and domain predicates in the FOL. Second, constraints are created from the remaining predicates in the FOL. Abductive reasoning needs an additional step that will be presented in Section §B.1.3.

Step 1: Variables and domains

The FOL representation of SFMs relies on $state(E,S)$ predicates. Each element constant E can have as many states S as its domain predicates indicate. Each possible state S is defined by a state constant in the FOL. To represent this information in a CSP, an *element variable*(e) is created for each element constant E. For each state that element can have, it is assigned a value number that is used to represent the state in the CSP. The domain for an element variable is defined by those values that represent the states it can have. So for the two kinds of elements defined in the SFMM the resulting variables and domains are:

- Features: A feature constant F_j in FOL maps into an element variable f_j in a $\{0,1\}$ domain. Each cardinal in the domain represents a potential state the element can have: selected ($f_j = 1$) or removed ($f_j = 0$).
- Root: The root constant only has a *selected* state, so its domain is reduced to $\{1\}$.
- Cardinality: A cardinality constant C_i in FOL maps into an element variable c_i in a domain containing each cardinal in the cardinality. So a <1..3> cardinality maps into a $\{1,2,3\}$ domain. Non-valid cardinals within the range between 0 and the number of child features are removed from the cardinality variable domain so constraints are not needed to explicitly avoid non-valid cardinals. For the mapping that we propose, any number can be assigned for a cardinal since the number itself is not used for any calculation. However we use domains whose values represent the value for the cardinal for clarity.

A domain in a CSP has an implicit exclusive-or relationship. So a feature variable f_j whose domain is $\{0,1\}$ can only have either value 0 or 1 in a solution. If at a cer-

tain time the feature variable is assigned a value such as $f_j = 1$ then its domain is immediately reduced to $\{1\}$ and value 0 is discarded. This is the reason why domain predicates are not needed in a CSP and are left out of the mapping.

Step 2: Constraints

The predicates in the KB of the FOL constrain the element states. Symmetrically, constraints in the CSP represent how the domains are constrained. Since there is an equivalence between *state* predicates and domain values, it is easy to find a correspondence of FOL predicates and constraints. A predicate $state(E, V)$ maps into a constraint $e = v$ while a predicate $\neg state(E, V)$ maps into a constraint $e <> v$. The semantics for each relationship can be represented in terms of constraints applying this simple substitution. Since CSPs make no difference between syntax and semantics, each appearance of a relationship predicate such as *mandatory* or *optional*, must be substituted by the corresponding semantics. So the general mapping from FOL to CSP is summarised in Table §B.1. Table §B.2 shows the CSP that results from applying the above mapping to the FOL in Table §7.1 that describes the example SFM.

B.1.2 Deduction in constraint programming

Once a SFM is represented as a CSP, three operations can be used to obtain information from the problem using deduction: satisfiability, propagation and obtain the solutions. Each of them is related to three query operations in the AASFM that we analyse next:

- Validation: A CSP is satisfiable if there exist at least one solution. That is, it is possible to find an assignment of values to variables such that it satisfies all the constraints. For a SFM, a solution is an assignment of states to elements in the model, either features or cardinalities. If the CSP is satisfiable it means that it is possible to find valid values for each feature and cardinality in the SFM. So the validation operation can be implemented for a SFM using CSP satisfaction. If the CSP is satisfiable, then the SFM is valid; otherwise the SFM is invalid. We are conscious that validation is not defined as a core operation. However this operation offers a better performance than product listing in which the compound operation relies on.

- Propagation: in constraint programming, constraint propagation techniques are methods to produce equivalent CSPs which are simpler to solve. In general, con-

FOL to CSP Mapping

Variables and Domains

FOL Constant	CSP Variable	Domain
$F_1, ..., F_n$	$f_1, f_2, ..., f_n$	$\{0,1\}$
F_{Root}	f_{Root}	$\{1\}$
$C_1, C_2, ..., C_m$	$c_1, ..., c_m$	Cardinals in cardinality

State predicates as constraints

Relationship Predicate	Constraint
$state(F_i, sel), \neg state(F_i, sel)$	$f_i = 1, f_i <> 1$
$state(F_i, rem), \neg state(F_i, rem)$	$f_i = 0, f_i <> 0$
$state(C_i, v), \neg state(C_i, v)$	$c_i = v, c_i <> v$

FOL relationships to constraints

$mandatory(F_p, F_c)$	$f_p = f_c$
$optional(F_p, F_c)$	$f_c \Rightarrow f_p$
$set_3(F_p, C_j, F_{c_1}, ..., F_{c_n})$	$f_p = 1 \Leftrightarrow (c_j = 0 \Leftrightarrow (f_{c_1} = 0 \wedge ... \wedge f_{c_n} = 0)) \wedge$ $(c_j = 1 \Leftrightarrow (f_{c_1} = 1 \oplus ... \oplus f_{c_n} = 1)) \wedge$ $(c_j = 2 \Leftrightarrow (...)) \wedge$ ... $(c_j = n \Leftrightarrow (f_{c_1} = 1 \wedge ... \wedge f_{c_n} = 1))$ $\wedge f_p = 0 \Rightarrow \bigwedge_i f_{c_i} = 0$
$depends(F_a, F_b)$	$f_a = 1 \Rightarrow f_b = 1$
$excludes(F_a, F_b)$	$f_a = 1 \Leftrightarrow f_b = 0$

FOL configurations to constraints

$choose(E, V)$	$e = v$
$discard(E, V)$	$e <> v$

Table B.1: Mapping Explanatory Operations into a CSP

straint propagation removes those values in the domain that are not possible for any solution given the constraints. So, if a feature is core such as B in the example SFM, its domain will be reduced to $f_B \in \{1\}$ as a result of constraint propagation due to $f_{Root} = f_B$ constraint. There are many propagation techniques, being arc-consistency generally and AC-3 [58] specifically the most used ones.

Propagation in SFMs can be implemented by means of constraint propagation. The variable domains that result from CSP propagation are a subset of the origi-

Example SFM as a CSP

Variables

FOL Constant	**CSP Variable**
F_A, F_B, F_C, F_D, F_E	f_A, f_B, f_C, f_D, f_E
F_{Root}	f_{Root}
C_1	c_1

Domains

FOL Domain	**CSP Domain**
$state(F_{Root}, selected)$	$f_{Root} \in \{0,1\}$
$state(F_A, sel) \oplus state(F_A, rem)$	$f_A \in \{1\}$
...	...
$(state(C_1,1) \vee state(C_1,2))$ $\wedge \neg state(C_1,3)$	$c_1 \in \{1,2\}$

Constraints

$optional(F_{Root}, F_A)$	$f_A \Rightarrow f_{root}$
$mandatory(F_{Root}, F_B)$	$f_{Root} = f_B$
$set_3(F_B, C_1, F_C, F_D, F_E)$	$f_B = 1 \Leftrightarrow ($ $c_1 = 0 \Leftrightarrow ($ $f_C = 0 \wedge f_D = 0 \wedge f_E = 0)$ $c_1 = 1 \Leftrightarrow ($ $f_C = 1 \oplus f_D = 1 \oplus f_E = 1) \wedge$ $c_1 = 2 \Leftrightarrow ($ $(f_C = 1 \wedge f_D = 1) \vee$ $(f_C = 1 \wedge f_E = 1) \vee$ $(f_D = 1 \wedge f_E = 1)) \wedge$ $c_1 = 3 \Leftrightarrow ($ $f_C = 1 \wedge f_D = 1 \wedge f_E = 1))$ $\wedge f_B = 0 \Rightarrow ($ $f_C = 0 \wedge$ $f_D = 0 \wedge f_E = 0)$
$depends(F_D, F_E)$	$f_D = 1 \Rightarrow f_E = 1$
$excludes(F_A, F_C)$	$f_A = 1 \Leftrightarrow f_C = 0$
$choose(F_C, sel)$	$f_C = 1$

Table B.2: Example SFM as a CSP

nal domains. The discarded values in the domains represent the states that must be discarded in the SFM by means of automatic configurations.Under some circumstances, propagation can produce empty domains which indicates an inconsistent CSP where it is impossible to find any solution. That is the case of a void SFM for example where the root feature will have an empty domain after propagation.

If constraint propagation is executed for the CSP in Table §B.2, the resulting domains are $f_{Root} \in \{1\}$, $f_A \in \{0\}$, $f_B \in \{1\}$, $f_C \in \{1\}$, $f_D \in \{0,1\}$, $f_E \in \{0,1\}$. As a result of the selection of feature `C`, its parent feature `B` must be selected and feature `A` removed.

- Product Listing: A solution of a CSP is an assignment for all the variables in the problem. if $f_i \mapsto 0$ for a solution it indicates that feature `Fi` is removed; $f_i \mapsto 1$ sets a feature `Fi` as selected. Solving the CSP produces as many solutions as products a SFM defines. Each solution can be mapped to products since each variable corresponds to a feature constant in FOL and it corresponds to a feature in the SFM. The assignments obtained for cardinality variables are ignored since they are not relevant for this operation. The solutions for the CSP in Table §B.2 and the correspondence with products are the following:

$$\begin{aligned} f_{Root} \mapsto 1, f_A \mapsto 1, f_B \mapsto 1, f_C \mapsto 1, f_D \mapsto 0, f_E \mapsto 1 &\Rightarrow P_1 = \{Root, A, B, C, E\} \\ f_{Root} \mapsto 1, f_A \mapsto 0, f_B \mapsto 1, f_C \mapsto 1, f_D \mapsto 0, f_E \mapsto 0 &\Rightarrow P_2 = \{Root, B, C\} \\ f_{Root} \mapsto 1, f_A \mapsto 0, f_B \mapsto 1, f_C \mapsto 1, f_D \mapsto 0, f_E \mapsto 1 &\Rightarrow P_3 = \{Root, B, C, E\} \end{aligned}$$

Note that all the obtained products contain the feature `C` since the configuration imposes it. Mapping FMs to CSPs is a common approach in the AAFM. The mapping that we propose in this Chapter is very similar to the one proposed in [10]. The main difference resides in the representation of cardinalities and the way set-relationships are represented. This way of representing cardinalities will allow the treatment of cardinals as first-class elements even in the CSP which was one of the solutions that introduces the AASFM.

B.1.3 Abduction in constraint programming

Dechter and Dechter [32] propose interpreting an abduction problem as a *Constraint Optimisation Problem (COP)*. This approach is inspired in that work to implement abduction in our CSPs representation. First, the KB represented by constraints is split into facts and hypotheses, resulting a new CSP. Second, an optimisation function is defined to transform a CSP into a COP able to solve abduction.

Step 3: Separating Facts, Hypotheses and Observations

Assumption Variables (A_i) are introduced to distinguish hypotheses from fact and observations. They take values in domain $\{0,1\}$. For each predicate in the hypotheses set H, an assumption variable is created. Let us consider that a predicate is transformed

into a constraint C in the CSP. That constraint is substituted by a constraint in the form $A_i = 0 \Leftrightarrow C$ so that if $A_i \mapsto 0$ then the constraint C must be satisfied; if $A_i \mapsto 1$ then the constraint C must be violated. So a CSP solver can play with these variables to activate or deactivate the constraints as needed. In our case, we use them to de/activate relationships in a potential explanation so all the subsets of hypotheses can be checked if are consistent or not.

The way the KB is split depends on the kind of explanatory analysis to perform. Table §B.3 applies this separation to all the relationship constraints in the CSP in Table §B.2 to perform relationships explanatory analysis.

For configurations explanatory analysis, the hypotheses set is formed by configuration constraints, so it results a different separation of knowledge. Table §B.4 shows the result for the example SFM with a configuration that sets features `C`, `D` and `E`, which is an invalid configuration. Note that the correspondence between assumption variables and the relationships or configuration elements that they affect is stored so the obtained results in abduction can be mapped back to the respective elements in the SFM.

Solving the abduction problem

A solution for the so-obtained CSP is an assignment of values to element and assumption variables. Assigning values to assumption variables allows to configure a subset of hypotheses that is taken into account to solve the problem. If an assumption variable is assigned $A_i \mapsto 1$ then its constraint must be violated and set aside to solve the CSP. In case $A_i \mapsto 0$, the constraint must be satisfied in the assignment of a solution. So playing with the values of assumption variables it is possible to introduce hypotheses into the problem or leaving them out.

In abduction only assumption variables are relevant for a solution. In our case we want to obtain the relationship or configuration constraints that are in an explanation but element variables values are irrelevant for this purpose. So element variables are left out of consideration for our purpose and only assumption variables are part of the solutions.

With the above definition of the problem, all possible combinations of assumption variables are obtained. However we are interested in obtaining the minimal subset of hypotheses that produces an inconsistency, i.e. whose assumption variable is $A_i \mapsto 1$. If we consider $minimal_2$ criterion, searching for those explanations with the minimal number of violated constraints. Since a violated constraint is marked giving a value 1

Example SFM as a CSP for Relationship Explanation

Variables and Domains

Constant in FOL	**Variable**	**Domain**
F_A, F_B, F_C, F_D, F_E	f_A, f_B, f_C, f_D, f_E	$\{0,1\}$
F_{Root}	f_{Root}	$\{1\}$
C_1	c_1	$\{1,2\}$
Relationship	**Variable**	**Domain**
$optional(F_{Root}, F_A)$	A_1	$\{0,1\}$
$mandatory(F_{Root}, F_B)$	A_2	$\{0,1\}$
$set_3(F_B, C_1, F_C, F_D, F_E)$	A_3	$\{0,1\}$
$depends(F_D, F_E)$	A_4	$\{0,1\}$
$excludes(F_A, F_C)$	A_5	$\{0,1\}$

Constraints

$optional(F_{Root}, F_A)$	$A_1 = 0 \Leftrightarrow (f_A \Rightarrow f_{root})$
$mandatory(F_{Root}, F_B)$	$A_2 = 0 \Leftrightarrow (f_{Root} = f_B)$
$set_3(F_B, C_1, F_C, F_D, F_E)$	$A_3 = 0 \Leftrightarrow (f_B = 1 \Leftrightarrow (\ c_1 = 0 \Leftrightarrow (f_C = 0 \wedge f_D = 0 \wedge f_E = 0) \wedge$ $c_1 = 1 \Leftrightarrow (f_C = 1 \oplus f_D = 1 \oplus f_E = 1) \wedge$ $c_1 = 2 \Leftrightarrow ((f_C = 1 \wedge f_D = 1) \vee$ $(f_C = 1 \wedge f_E = 1) \vee$ $(f_D = 1 \wedge f_E = 1)) \wedge$ $c_1 = 3 \Leftrightarrow (f_C = 1 \wedge f_D = 1 \wedge f_E = 1))$ $\wedge f_B = 0 \Rightarrow (f_C = 0 \wedge \quad f_D = 0 \wedge f_E = 0))$
$depends(F_D, F_E)$	$A_4 = 0 \Leftrightarrow (f_D = 1 \Rightarrow f_E = 1)$
$excludes(F_A, F_C)$	$A_5 = 0 \Leftrightarrow (f_A = 1 \Leftrightarrow f_C = 0)$

Table B.3: Example SFM as a CSP for relationship explanation

for its assumption variable in a solution, the goal is to minimise the number of assumption variables such that $A_i \mapsto 1$. So the optimisation function for the COP is defined as follows:

$$O_{exp} = min(\sum_i a_i)$$

For example, B is a core feature in the example SFM and we want to know why. To solve the operation, the observation $\{choose(F_B, rem)\}$ is set in FOL, that is mapped into a constraint $\{f_B = 0\}$ that is added to the CSP in Table §B.3. A solution that minimises the optimisation function is $\{A_1 \mapsto 0, A_2 \mapsto 1, A_3 \mapsto 0, A_4 \mapsto 0, A_5 \mapsto 0, ...\}$. It indicates

Example SFM as a CSP for Configuration Explanation

Variables and Domains

Constant in FOL	**Variable**	**Domain**
F_A, F_B, F_C, F_D, F_E	f_A, f_B, f_C, f_D, f_E	$\{0,1\}$
F_{Root}	f_{Root}	$\{1\}$
C_1	c_1	$\{1,2\}$
Configuration Elem.	**Variable**	**Domain**
$choose(F_C, sel)$	A_1	$\{0,1\}$
$choose(F_D, sel)$	A_2	$\{0,1\}$
$choose(F_E, sel)$	A_3	$\{0,1\}$

Constraints

$optional(F_{Root}, F_A)$	$f_A \Rightarrow f_{root}$
$mandatory(F_{Root}, F_B)$	$f_{Root} = f_B$
$set_3(F_B, C_1, F_C, F_D, F_E)$	$f_B = 1 \Leftrightarrow (\ c_1 = 1 \Leftrightarrow (\ f_C = 1 \oplus f_D = 1 \oplus f_E = 1) \wedge$ $c_1 = 2 \Leftrightarrow (\ (f_C = 1 \wedge f_D = 1) \vee$ $(f_C = 1 \wedge f_E = 1) \vee$ $(f_D = 1 \wedge f_E = 1)) \wedge$ $c_1 = 3 \Leftrightarrow (\ f_C = 1 \wedge f_D = 1 \wedge f_E = 1))$ $\wedge f_B = 0 \Leftrightarrow (\ f_C = 0 \wedge \ f_D = 0 \wedge f_E = 0)$
$depends(F_D, F_E)$	$f_D = 1 \Rightarrow f_E = 1$
$excludes(F_A, F_C)$	$f_A = 1 \Leftrightarrow f_C = 0$
$choose(F_C, sel)$	$A_1 = 0 \Leftrightarrow f_C = 1$
$choose(F_D, sel)$	$A_2 = 0 \Leftrightarrow f_D = 1$
$choose(F_E, sel)$	$A_3 = 0 \Leftrightarrow f_E = 1$

Table B.4: Example SFM as a CSP for configuration explanation

that the problem is satisfiable if all the constraints are considered but the mandatory constraint linked to A_2. So an explanation for the abduction problem is $\{A_2\}$ which corresponds to $\{mandatory(F_{Root}, F_B)\}$ in FOL and the mandatory relationship in the SFM.

Note that the same set of assumption variables can appear for several solutions whose value for O_{exp} is the same due to different assignments for the set of feature and cardinality variables. So a post process is needed to remove element variables and eliminate duplicated solutions.

For a configuration explanatory operations the procedure is very similar. A configuration selecting `C`, `D` and `E` features at the same time is clearly invalid for the example SFM. To determine why such configuration is not valid the mapping in Table §B.4 can be used. Two solutions are obtained from the COP:

$$\{A_1 \mapsto 1, A_2 \mapsto 0, A_3 \mapsto 0\} \rightarrow \{choose(F_C, sel)\} \rightarrow \text{select C}$$
$$\{A_1 \mapsto 0, A_2 \mapsto 1, A_3 \mapsto 0\} \rightarrow \{choose(F_D, sel)\} \rightarrow \text{select D}$$

So either `C` or `D` feature selection has to be eliminated from the configuration to be valid.

B.2 DEFAULT LOGIC

When we model our relevant world in a KB, predicates represent what we know that is true and false. Anything that is not within a KB is just unknown. This assertion is known as *Open World Assumption*(OWA). However deductive reasoning cannot reach for conclusions taking into account ignorance as entailment cannot handle it. The most frequent solution is known as the *Closed World Assumption (CWA)* [73]. CWA assumes that a predicate P is false if we don't know if P is true, i.e. it is not possible to infer P from KB ($KB \not\models P \Rightarrow KB \models \neg P$). CWA may be summarised as *"what is not known to be true is just false"*.

CWA is useful in contexts where it can be ensured that a KB is complete as it gathers all the needed information. CWA is applicable for example to a database for the workers in a company -no person that is not in the database, may be a worker- or to a flight connection database -any connection that is not in the database is not available-. However well-bounded databases are not the most frequent situation. In the remaining cases, compiling all the available information into a KB is an infeasible problem. So a KB must be defined taking into account that new knowledge will appear in the future and may even invalidate previous knowledge.

In CWA, conclusions ($Concl$) entailed from the KB ($KB \models Concl$) cannot be contradicted by new knowledge(NK) since it will contradict the knowledge in the KB itself, i.e. $Concl \cup NK \models false \Rightarrow KB \cup NK \models false$. It means that anything that is concluded to be true or false will remain being true or false in the future. This is known as the *monotonicity of entailment* and logics with such a property are known as *monotonic logics*.

Default Logic (DL) [74] represents the knowledge in terms of facts and default rules

(or just defaults) which are described in terms of FOL. Default rules introduce new knowledge that might arise when certain assumptions are true. If any of those assumptions are contradicted later, that knowledge is invalidated. Therefore, any conclusion made from a prior knowledge can be contradicted by new knowledge, which makes DL a non-monotonic logic. A default has the following syntax:

$$\frac{pre(x_1,\cdots,x_i) : justif(y_1,\cdots,y_j)}{concl(z_1,\cdots,z_k)}$$

It means that whenever pre_i is true and $justif$ is not false, then $concl$ can be assumed to be true. pre, $justif$ and $concl$ predicates are described in terms of FOL. For example, the following rule indicates that a bird flies by default unless it is specifically contradicted:

$$\frac{bird(x) : fly(x)}{fly(x)}$$

So $bird(gull)$ predicate will lead to assuming that $fly(gull)$. And so the same for $bird(penguin)$. However, if we know that $\neg fly(penguin)$ the previous assumption $fly(penguin)$ is invalidated.

Let F be a set of facts expressed in terms of FOL; D a set of default rules and let $DL_0 =< F, D >$ be a default logic. The default reasoning process generates subsequent new logics applying a default rule for each of them until no defaults can be applied as follows:

$$S_1 = F \cup c_i(g_z) \quad \text{s.t} \quad d_i \in D, d_i \equiv \frac{p_i(x) : j_i(y)}{c_i(z)}, F \models p_i(g_x), F \not\models \neg j_i(g_y)$$

$$S_2 = F \cup c_i(g_z) \cup c_j(g'_z) \quad \text{s.t} \quad d_j \in D, d_j \equiv \frac{p_j(x') : j_j(y')}{c_j(z')}, F \models p_j(g'_x), F \not\models \neg j_j(g'_y)$$

Note that default rules can contain variables. However they cannot be used for default reasoning but ground terms instead (constants and functions). g_x, g_y and g_z represent a set of ground terms so that ground formulas are obtained as a result and therefore incorporated to the set of facts in each step. We call S_1 and S_2 *scenarios* for the logic, such that several scenarios can be obtained from a logic. A scenario such that no default applies (i.e. the scenario is a fixed point for the reasoning function) is known as an *extension* for the logic.

DL also divides the knowledge in terms of facts and hypotheses, represented as default rules. It makes DL suitable for abduction. So we have to transform the tuple

$< F, H, O >$ obtained from the mapping of SFMs to FOL, into a DL problem. F are true under any circumstance so they become facts in the default logic. The set of hypotheses H is transformed into a set of defaults in the following form:

$$Defaults(H) = \{\frac{: relationship(F_k, \cdots, F_j)}{relationship(F_k, \cdots, F_j)} | relationship(F_k, \cdots, F_j) \in H\}$$

Such that *relationship* represents any predicate such as *mandatory*, *optional*, etc. in the set of hypotheses. Table §B.5 shows the default logic that results from mapping the example SFM in FOL to perform relationships explanatory analysis. A default expressed in this form indicates that a relationship may be valid for a SFM if it is consistent with the facts and the relationships that were previously introduced. Introducing relationships one by one together with the set of facts builds a set of valid scenarios or explanations for our problem. From all those scenarios we only seek for those which explain or entail the observation. These scenarios are the explanations for the explanatory operation. DL can solve an abduction problem if we search for all the scenarios for a given default logic such that each of them explains (entails) the observation for the corresponding operation. So an abduction reasoner can be implemented by means of DL in the following terms:

$$Exp(F, H, O) = \{S | S \models O, S \in DL(F, Defaults(H))\}$$

Minimality criteria can be used to obtain minimal explanations in the same way that is proposed for the FOL approach.

For configurations explanatory operations, the set of defaults is composed of *choose* and *discard* predicates instead of relationship predicates. Table §B.6 shows the default logic that can be used to perform abduction on configurations. The resolution of the problem remains the same than for relationships explanatory operations.

B.3 OTHER IMPLEMENTATIONS

Besides default logic and constraint programming there are many other techniques that are used in the literature to solve abduction problems. We have compiled the most used techniques to foresee future possible implementations of explanatory analysis:

- **Propositional Logic**: Eliyahu and Dechter [38] propose a mapping from default

Default Logic for Relationships Explanation

Facts

$state(F_{root},sel)$
$state(F_A,sel) \oplus state(F_A,rem)$
$state(F_B,sel) \oplus state(F_B,rem)$
$state(F_C,sel) \oplus state(F_C,rem)$
$state(F_D,sel) \oplus state(F_D,rem)$
$state(F_E,sel) \oplus state(F_E,rem)$
$(state(C_1,1) \vee state(C_1,2)) \wedge \neg state(C_1,3)$
$choose(F_C,sel)$

Defaults

$\frac{:optional(F_{root},F_A)}{optional(F_{root},F_A)}$ $\frac{:mandatory(F_{root},F_B)}{mandatory(F_{root},F_B)}$
$\frac{:set_3(F_B,C_1,F_C,F_D,F_E)}{set_3(F_B,C_1,F_C,F_D,F_E)}$ $\frac{:depends(F_D,F_E)}{depends(F_D,F_E)}$
$\frac{:excludes(F_A,F_C)}{excludes(F_A,F_C)}$

Table B.5: Example SFM in terms of default logic for relationships explanation

logic to propositional logic. Following this mapping we might transform an abduction problem into a propositional satisfiability (SAT) problem. SAT problems can determine if a propositional formula is true or false, which can be used for SFM validation. Propositional logics are a common approach to solve query operations [5, 12, 64, 84]. Current version of FAMA Framework solves explanatory analysis for errors, relying on SAT4j [15], a SAT solver.

- **Prolog Metainterpreters**: Prolog [50] relies on definite clause logic to represent a KB. Prolog provides by default a resolution engine to solve deductive queries. However it is possible to define meta-interpreters that slightly change the way Prolog reasons about a knowledge base. Flach describes a meta-interpreter[40, page 159-162] to support abduction and diagnosis. The metainterpreter distinguishes between facts and hypotheses by means of an *abducible* predicate which is used to point out hypotheses. An abduction reasoning process is launched building a query on *abduce* predicate that searches for the set of *abducible* predicates that satisfy a predicate. Since facts and defaults use first-order predicates, they can be translated into clausal logic following the process described in [40, page 38-41]. Prolog has been used to approach some query operations in [53,

Default Logic for Configuration Explanation

Facts

$state(F_{root}, sel)$
$state(F_A, sel) \oplus state(F_A, rem)$
$state(F_B, sel) \oplus state(F_B, rem)$
$state(F_C, sel) \oplus state(F_C, rem)$
$state(F_D, sel) \oplus state(F_D, rem)$
$state(F_E, sel) \oplus state(F_E, rem)$
$(state(C_1, 1) \vee state(C_1, 2)) \wedge \neg state(C_1, 3)$
$optional(F_{Root}, F_A)$
$mandatory(F_{Root}, F_B)$
$set_3(F_B, C_1, F_C, F_D, F_E)$
$depends(F_D, F_E)$
$excludes(F_A, F_C)$

Defaults

$\frac{:choose(E,V)}{choose(E,V)}$
$\frac{:discard(E,V)}{discard(E,V)}$

Table B.6: Example SFM in terms of default logic for configuration explanation

page 70] and [34].

- **Truth Maintenance Systems (TMS)**: Jon Doyle introduces TMS in 1979 [35] as a way to keep track on the conclusions that are made during a reasoning process and the justifications they rely on. Whenever a new predicate introduces a contradiction, justifications are used to restore the consistency of the problem, enabling undoing decisions. TMS are data-structures rather than a way of reasoning, so they are commonly combined with logic solvers to support abductive reasoning in particular and non-monotonic reasoning in general[25]. Doyle's TMS are considered as justification-based TMS (JTMS) as they rely on textual justifications to keep a record on the decisions that are made. Logic TMS (LTMS) arises as an evolution of JTMS that supports logic predicates as justifications rather than text. If we use a TMS for default reasoning, it can only store one possible scenario at a given time. So if there are more than one default that can be applied at a time, the TMS is only able to keep track of one of the resulting scenarios. To solve this

limitation, De Kleer [31] proposes Assumption-Based Truth Maintenance System (ATMS) that are able to keep track on several scenarios at the same time but are unable to support logic predicates. Check [60] for a complete review on the state of the art in TMS. Junker proposes in [52] using JTMS to reason about default logics, that can be used to inspire t he integration of TMS in other of the here proposed implementations. A preliminary approach to use LTMS to obtain explanations in void SFMs was presented in [5].

- **Temporal Logics**: Temporal logics allow to represent the evolution in time of a system that can be described in terms of a logic. A translation from default logic into temporal partial logic is proposed by Engelfriet and Treur [51]. This translation interprets the abductive reasoning process as an evolution in time of knowledge.

C

First-order logics

A first-order predicate logic uses *first-order languages* to represent the knowledge and to reason about it. As any language, a first-order language is defined in terms of syntax and semantics. The syntax contains the following symbols [61]:

- The propositional connectives $\neg$ and $\Rightarrow$ ($\wedge$, $\vee$ and $\Leftrightarrow$ can be represented as a combination of $\neg$ and $\Rightarrow$).
- Commas and parenthesis.
- The *universal quantifier* $\forall$ (*existential quantifier* $\exists$ can by represented by $\forall$)
- A denumerable set of individual *variables* commonly represented by lower-case letters $x, y, z, ..$
- A denumerable set of individual *constants* represented by lower-case letters $a, b, c, ...$
- A denumerable set of *function* letters represented by f_k^n where k is an index and n its arity or number of arguments.
- A non-empty set of *predicate* letters represented by $A_k^n(...)$.

Definition C.1.
In FOL a *term* is either:

1. A variable or constant.
2. A function letter $f_k^n(t_1, t_2, ..., f_n)$ where $t_1, t_2, ..., t_n$ are terms.
3. Nothing else (a predicate for example) is a term.

A term which contains no variables is a *ground term*.

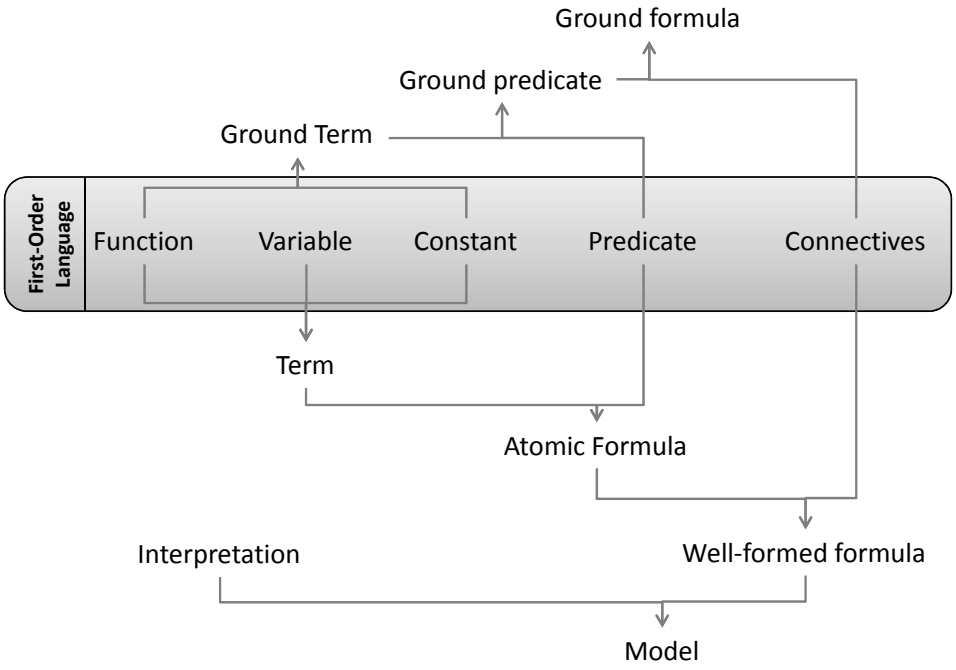


Figure C.1: Elements of First Order Languages and their Relationships

Definition C.2.
An *atomic formula* is a predicate $A_k^n(t_1, t_2, ..., t_n)$ where $t_1, t_2, ..., t_n$ are terms. If $t_1, t_2, ..., t_n$ are ground terms then $A_k^n(t_1, t_2, ..., t_n)$ is a ground predicate.

Definition C.3.
A *well-formed formula*(wff) in FOL is defined as follows:

1. Every atomic formula is a wff.
2. Let B and C be wff and y a variable then $(\neg B)$, $(A \Rightarrow B)$ and $((\forall y)B)$ are wffs.
3. Nothing else is a wff.

Definition C.4.
A variable is said to be *bound* in a wff if it is part of a quantifier $(\forall x)$ or its scope. Otherwise, a variable is *free* for that wff. Taking $((\forall x)A_1^2(x, y)$ as an example, x is a bound variable while y is a free variable. A *closed wff* is the one that contains no free variables.

A wff is a *ground formula* if it contains neither free nor bound variable. Therefore quantifiers are not allowed in ground formulas.

Example Let a,b and c be constants, x a variable, A_1^2 a predicate and f_1^2 a function. Three examples of wffs are shown below:

$$\begin{array}{ll} A_1^2(a,x) & (x \text{ is a free variable}) \\ ((\forall x)A_1^2(a,x)) & (x \text{ is a bound variable}) \\ A_1^2(a,f_1^2(b,c)) & (A_1^2 \text{ is a ground formula}) \end{array}$$

In predicate calculus a statement needs an assignment of truth values to its letter statements to determine if it is true or false. Likewise, a wff has no meaning unless an *interpretation* or semantics is given to its symbols. An interpretation M of a language $\mathcal{L}$ is composed by:

1. A non-empty set D called the domain of M.
2. A predicate letter $A_k^n(t_1,t_2,...,t_n)$ is mapped into a relation over D^n where for each t_i it is assigned an element in D, i.e. a predicate defines a subset of D^n.
3. A function letter $f_k^n(t_1,t_2,...,t_n)$ is mapped into a function $f_k^i : D^n \to D$ such that an element in D is assigned for each tuple $t_1,t_2,...,t_n$.
4. For each constant a_1,a_2,a_3... an interpretation assigns a fixed element $(a)^M$ of D.

An interpretation introduces a semantics to predicates and functions and assigns values to constants.

Example a wff $\mathcal{W}_1 \equiv P^1(a) \wedge (\forall x)Q^2(f^2(x,a),x)$ has no meaning without a semantics. Consider the following interpretation M_1:

- **Domain**: $\mathbb{N}_0$.
- **Predicates**[1]: $P^1(y) \equiv y = 0$; $Q^2(y,z) \equiv y = z$.

[1]Solving equality expressions in FOL has many implications in the reasoning procedure. For the sake of simplicity, we consider that $x = y$ if both variables or constants map into the same value in the interpretation domain.

- **Functions**: $f^2(y,z) = y + z$.
- **Constants**: $a \mapsto 0$.

Given the above interpretation $\mathcal{W}_1$ defines the identity element property for natural numbers.

D

Acronyms

AAFM
Automated Analysis of Feature Models.

AASFM
Automated Analysis of Stateful Feature Models.

AP
Abduction Problem.

ATL
Atlas Transformation Language.

BDD
Binary Decision Diagram.

BPMN
Business Process Model and Notation.

CBFM
Cardinality-Based Feature Model.

CM
Configuration Model.

COP
Constraint Optimisation Problem.

CSP
Constraint Satisfaction Problem.

CWA
Closed World Assumption.

DAP
Deductive and Abductive Problem.

DP
Deduction Problem.

EFM
Extended Feature Model.

FM
Feature Model.

FOL
First Order Logic.

KB
Knowledge Base.

MDE
Model-driven Engineering.

SFM
Stateful Feature Model.

SFMM
Stateful Feature Metamodel.

SFMP
Stateful Feature Model Problem.

SHS
Smart Home System.

SLA
Service-Level Agreement.

SPL
Software Product Line.

StaPLe
Stateful Predicate Logic.

STEAm
: STateful fEature model Analyser.

BIBLIOGRAPHY

[1] T. O. Alliance. About the OSGi Service Platform. http://www.osgi.org/documents/osgi_technology/osgi-sp-overview.pdf, July 2004. (page 135).

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 0-521-78176-0. (page 27).

[3] R. Bachmeyer and H. Delugach. A conceptual graph approach to feature modeling. In U. Priss, S. Polovina, and R. Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications*, volume 4604 of *Lecture Notes in Computer Science*, pages 179–191. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-73680-6. (page 28).

[4] E. Bagheri, T. D. Noia, D. Gasevic, and A. Ragone. Formalizing interactive staged feature model configuration. *Journal of Software: Evolution and Process*, 24(4):375–400, 2012. doi: 10.1002/smr.534. (pages 30, 31).

[5] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference, LNCS 3714*, pages 7–20, 2005. (pages 4, 5, 27, 31, 33, 189, 191).

[6] D. Batory. Ahead tool suite. http://www.cs.utexas.edu/users/schwartz/ATS.html, 2006. (pages 5, 33).

[7] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, 49(12):45–47, December 2006. (pages 5, 22, 28).

[8] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and P. F. Patel-Schneider. *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2004. URL http://www.w3.org/TR/owl-ref/. (page 27).

[9] D. Benavides. *On the Automated Analysis of Software Product Lines Using Feature Models. A framework for developing automated tool support.* PhD thesis, University of Seville, http://www.lsi.us.es/~dbc/dbc_archivos/pubs/benavides07-phd.pdf, 2007. (page 5).

[10] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005. ISSN 0302-9743. (pages 5, 6, 18, 27, 54, 182).

[11] D. Benavides, S. Trujillo, and P. Trinidad. On the modularization of feature models. In *First European Workshop on Model Transformation*, September 2005. (page 134).

[12] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006. (pages 27, 28, 189).

[13] D. Benavides, A. Metzger, and U. W. Eisenecker, editors. *Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, January 28-30, 2009. Proceedings*, volume 29 of *ICB Research Report*, Essen, Germany, 2009. Universität Duisburg-Essen. (pages 203, 209).

[14] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 9 2010. (pages 5, 6, 22, 23, 26, 28, 55, 78, 91, 114, 177).

[15] D. L. Berre. Sat4j: The satisfiability library for java (www.sat4j.org). http://www.sat4j.org, 2004. (pages 26, 134, 148, 189).

[16] Y. Bontemps, P. Heymans, P. Schobbens, and J. Trigaux. The semantics of foda feature diagrams. In *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004. (page 5).

[17] J. F. Carlos Cetina, Vicente Pelechano. Moskitt feature modeler. http://www.pros.upv.es/mfm/, 2008. (page 135).

[18] V. Cechticky, A.Pasetti, O. Rohlik, and W. Schaufelberger. Xml-based feature modelling. *LNCS, Software Reuse: Methods, Techniques and Tools: 8th ICSR 2004. Proceedings*, 3107:101–114, 2004. (page 28).

[19] C. Cetina, P. Trinidad, V. . Pelechano, and A. Ruiz-Cortés. An architectural discussion on dspl. In *2nd International Workshop on Dynamic Software Product Line (DSPL08)*, 2008. (page 167).

[20] C. Cetina, P. Trinidad, V. . Pelechano, and A. Ruiz-Cortés. Customisation along lifecycle of autonomic homes. In *3rd International Workshop on Dynamic Software Product Line (DSPL09)*, 2009. (page 52).

[21] C. Cetina, P. Trinidad, V. Pelechano, A. Ruiz-Cortés, and D. Benavides. Moskitt fm and fama fw: Taking feature models to the next level. In *XIV Jornadas de Ingeniería del Software y Bases de Datos*, pages 285–288, 2009. (page 134).

[22] A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In Benavides et al. [13], pages 51–60. (page 5).

[23] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, Aug. 2001. (pages 4, 14).

[24] I. C. S. Competition. Xml representation of constraint networks format xcsp 2.1. `http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf`, 2008. (pages 137, 146).

[25] A. Croker and V. Dhar. A problem-solver/tms architecture for general constraint satisfaction. Technical report, Information Systems Working Papers Series, 1988. (page 190).

[26] W. Cunningham. Episodes: A pattern language of competitive development. In J. Vlissides, editor, *Pattern Languages of Program Design 2*. Addison-Wesley, 1996. (page 162).

[27] K. Czarnecki and P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*, 2005. (page 27).

[28] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek. fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates. In *OOPSLA Companion*, pages 200–201. ACM, 2005. ISBN 1-59593-193-7. (page 28).

[29] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. (pages 15, 19, 29, 31).

[30] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005. (pages 29, 31).

[31] J. de Kleer. An assumption-based tms. *Artificial Intelligence*, 28(2):127–162, March 1986. doi: 10.1016/0004-3702(86)90080-9. URL http://dx.doi.org/10.1016/0004-3702(86)90080-9. (page 191).

[32] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *AAAI*, pages 37–42, 1988. (page 182).

[33] R. Dechter, A. Dechter, and J. Pearl. Optimization in constraint networks. In R. Oliver and J. Smith, editors, *Influence Diagrams, Belief Nets and Decision Analysis*, pages 411–425. John Wiley and Sons, 1990. (page 178).

[34] A. v. Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002. (page 190).

[35] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, November 1979. doi: 10.1016/0004-3702(79)90008-0. (page 190).

[36] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Flame: Fama formal framework (v 1.0). In *Transactions on Software Engineering and Methodology (submitted). Available as Technical Report at* http://www.isa.us.es/sites/default/files/FLAME-TR.pdf, Seville, Spain, March 2012. (page 5).

[37] A. O. Elfaki, S. Phon-Amnuaisuk, and C. K. Ho. Knowledge based method to validate feature models. In Thiel and Pohl [83], pages 217–225. ISBN 978-1-905952-06-9. (pages 5, 29, 30, 31).

[38] R. B. Eliyahu and R. Dechter. Default reasoning using classical logic. *Artificial Intelligence*, 84(1-2):113–150, 1996. (page 188).

[39] S. Fan and N. Zhang. Feature model based on description logics. In *Knowledge-Based Intelligent Information and Engineering Systems, 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006, Proceedings, Part II*, volume 4252 of *Lecture Notes in Computer Science*, pages 1144–1151. Springer, 2006. ISBN 3-540-46537-5. doi: 10.1007/11893004˙145. (page 5).

[40] P. Flach. *Simply Logical*. John Wiley, April 1994. ISBN 0-471-94152-2. (pages 27, 189).

[41] M. Folwer. *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0-201-48567-2. (page 167).

[42] J. Galindo, D. Benavides, and S. Segura. Debian packages repositories as software product line models. towards automated analysis. In *Proceeding of the First International Workshop on Automated Configuration and Tailoring of Applications (ACOTA)*, 2010. (page 134).

[43] J. García-Galán, P. Trinidad, and A. Ruiz-Cortés. Isa packager: A tool for spl deployment. In *Proceedings of the Fifth International Workshop on Variability Modelling of Software–intensive Systems (VAMOS)*, 2011. (page 156).

[44] J. García-Galán, O. Rana, P. Trinidad, and A. Ruiz-Cortés. Migrating to the cloud: a software product line based analysis (submitted). In *3rd Int'l Conference on Cloud Computing and Services Science (CLOSER 2013)*, 2012. (page 156).

[45] V. Haarslev and R. Moller. Description of the RACER system and its applications. In *Description Logics*, 2001. URL http://www.racer-systems.com. (page 27).

[46] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, 2008. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2008.123. (page 167).

[47] A. Hemakumar. Finding contradictions in feature models. In *First International Workshop on Analyses of Software Product Lines (ASPL'08)*, pages 183–190, 2008. (page 28).

[48] A. H. M. T. Hofstede and H. Proper. How to formalize it? formalization principles for information system development methods. *Information and Software Technology*, 40:519–540, 1998. (page 5).

[49] F. S. F. Inc. Gnu lesser general public licence version 3. http://www.gnu.org/copyleft/lesser.html, 2007. (page 133).

[50] ISO. Iso/iec 13211-1. international standard, information technology - programming languages - prolog - part 1: General core, 1995. (pages 27, 189).

[51] J. T. J. Engelfriet. A temporal model theory for default logic. In *ECSQARU '93: Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 91–96, London, UK, 1993. Springer-Verlag. ISBN 3-540-57395-X. (page 191).

[52] U. Junker and K. Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *AAAI-90 Proceedings*, 1990. (page 191).

[53] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990. (pages 4, 5, 7, 14, 27, 31, 33, 53, 54, 189).

[54] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. Featureide: A tool framework for feature-oriented software development. In *IEEE 31st International Conference on Software Engineering*, pages 611–614. IEEE, 2009. (page 28).

[55] M. Lang and C. Endler. Captainfeature. https://sourceforge.net/projects/captainfeature/, 2005. (page 28).

[56] C. Larman. *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, October 2004. ISBN 0131489062. (pages 162, 164).

[57] F. Lösch. *Optimization of variability in software product lines: a semi-automatic method for visualization, analysis, and restructuring of variability in software product lines*. PhD thesis, University of Stuttgart, 2008. (page 4).

[58] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8: 99–118, 1977. (pages 32, 180).

[59] M. Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer. (page 54).

[60] J. Martins. The truth, the whole truth, and nothing but the truth: An indexed bibliography to the literature of truth maintenance systems. *AI Mag.*, 11(5):7–25, January 1991. ISSN 0738-4602. (page 191).

[61] E. Mendelson. *Introduction to mathematical logic*. Discrete Mathematics and Its Applications Series. Chapman & Hall, 1997. ISBN 9780412808302. (page 193).

[62] M. Mendonça, M. Branco, and D. Cowan. S.p.l.o.t. - software product lines online tools. In *24th ACM SIGPLAN Conference on object oriented programming*

systems languages and applications - OOPSLA Companion, page 761, Orlando, Florida, USA, 10/2009 2009. ACM Press, ACM Press. ISBN 9781605587684. doi: 10.1145/1639950.1640002. (pages 5, 28, 147).

[63] M. Mendonça, D. D. Cowan, W. Malyk, and T. C. de Oliveira. Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. *JSW*, 3(2):69–82, 2008. doi: 10.4304/jsw.3.2.69-82. (pages 20, 30, 31).

[64] M. Mendonça, A. Wasowski, and K. Czarnecki. Sat-based analysis of feature models is easy. In Muthig and McGregor [65], pages 231–240. (pages 5, 27, 29, 189).

[65] D. Muthig and J. D. McGregor, editors. *Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24-28, 2009, Proceedings*, volume 446 of *ACM International Conference Proceeding Series*, 2009. ACM. (page 207).

[66] G. R. N. Jussien and X. Lorca. The choco constraint programming solver. In *Proceedings of workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, Paris,France, June 2003. (page 134).

[67] A. Nöhrer and A. Egyed. Conflict resolution strategies during product configuration. In D. Benavides, D. S. Batory, and P. Grünbacher, editors, *VaMoS*, volume 37 of *ICB-Research Report*, pages 107–114. Universität Duisburg-Essen, 2010. (pages 22, 30, 31).

[68] A. Nöhrer, A. Biere, and A. Egyed. A comparison of strategies for tolerating inconsistencies during decision-making. In E. S. de Almeida, C. Schwanninger, and D. Benavides, editors, *SPLC (1)*, pages 11–20. ACM, 2012. ISBN 978-1-4503-1094-9. (page 22).

[69] Obeo. Acceleo. http://www.acceleo.org/, 2006. (page 139).

[70] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, May 2007. ISBN 0978739256. (page 143).

[71] D. Poole, R. Goebel, and R. Aleliuna. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*. Springer-Verlag, 1987. (page 42).

[72] pure-systems GmbH. Variant management with pure::variants. Technical report, pure-systems GmbH, 2004. (pages 5, 28, 135).

[73] R. Reiter. On closed world data bases. In *Advances in Data Base Theory*, pages 55–76, San Francisco, CA, USA, 1978. Plemum Press. (page 186).

[74] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980. (page 186).

[75] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002. (pages 15, 18).

[76] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, Sept. 2006. (pages 15, 68).

[77] S. Segura, D. Benavides, P. Trinidad, A. Ruiz-Cortés, and J. Galindo. Betty. http://www.isa.us.es/betty, 2010. (pages 138, 147).

[78] S. Segura, J. Galindo, D. Benavides, J. Parejo, and A. Ruiz-Cortés. Betty: Benchmarking and testing on the automated analysis of feature models. In U. Eisenecker, S. Apel, and S. Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, pages 63–71, Leipzig, Germany, 2012. ACM. (pages 5, 147).

[79] B. L. Software. Gears. http://www.biglever.com, 2012. (page 5).

[80] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. Introducing pla at bosch gasoline systems: Experiences and practices. In R. L. Nord, editor, *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2004. ISBN 3-540-22918-3. (page 4).

[81] J. Sun, H. Zhang, Y. Li, and H. Wang. Formal semantics and verification for feature modeling. In *Proceedings of the ICECSS05*, 2005. (pages 32, 33, 53).

[82] R. Szymanek and K. Kuchcinski. Jacop: Java constraint programming (www.jacop.eu). http://www.jacop.eu/, 2001. (page 134).

[83] S. Thiel and K. Pohl, editors. *SPLC (2)*, 2008. Lero Int. Science Centre, University of Limerick, Ireland. ISBN 978-1-905952-06-9. (pages 204, 209, 210).

[84] T. Thüm, D. S. Batory, and C. Kästner. Reasoning about edits to feature models. In *ICSE*, pages 254–264. IEEE, 2009. ISBN 978-1-4244-3452-7. (page 189).

[85] P. Trinidad and A. Ruiz-Cortés. Abductive reasoning and automated analysis of feature models: How are they connected? In *3rd. International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, pages 145–153, Sevilla, Spain, Jan 2009. ICB Research Report N. 29. URL http://www.vamos-workshop.net/. (pages 5, 6, 26, 29, 31, 32, 33, 47, 53, 54, 78, 100, 104, 109, 110, 125, 128, 156).

[86] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. Isolated features detection in feature models. In *CAiSE Short Paper Proceedings*, 2006. (page 29).

[87] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and M. Toro. Explanations for agile feature models. In *Procceedings of the 1st International Workshop on Agile Product Line Engineering (APLE'06)*, 2006. (pages 28, 54).

[88] P. Trinidad, , A. Ruiz-Cortés, and J. P. na. Mapping feature models onto component models to build dynamic software product lines. In *1st International Workshop on Dynamic Software Product Line (DSPL07)*, 2007. (page 167).

[89] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008. doi: 10.1016/j.jss.2007.10.030. (pages 5, 6, 27, 32, 33, 54, 55, 156, 169).

[90] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In Thiel and Pohl [83]. ISBN 978-1-905952-06-9. (pages 5, 27, 28, 32, 33, 133).

[91] P. Trinidad, C. Müller, J. García-Galán, and A. Ruiz-Cortés. Building industry-ready tools: Fama framework and ada. In *Third International Workshop on Academic Software Development Tools and Techniques*, pages 160–173, 2010. (page 157).

[92] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995. ISBN 0-12-701610-4. (page 33).

[93] P. B. van den and I. Galvão. Analysis of feature models using generalised feature trees. In Benavides et al. [13], pages 29–35. (pages 29, 32, 33).

[94] T. von der Massen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, 2003. Springer Verlag. (page 28).

[95] T. von der Massen and H. Lichter. Deficiencies in feature models. In T. Mannisto and J. Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004. (page 28).

[96] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, November 2005. (pages 27, 32, 33, 53).

[97] J. Whaley. Javabdd. http://javabdd.sourceforge.net/, 2007. (pages 27, 134).

[98] J. White, D. Schmidt, D. B. P. Trinidad, and A. Ruiz-Cortes. Automated diagnosis of product-line configuration errors in feature models. In Thiel and Pohl [83]. ISBN 978-1-905952-06-9. (page 29).

[99] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094 – 1107, 2010. ISSN 0164-1212. doi: DOI: 10.1016/j.jss.2010.02.017. (pages 5, 6, 20, 22, 27, 29, 31, 52, 53, 54, 156, 167).

[100] L. A. Zaid, F. Kleinermann, and O. D. Troyer. Applying semantic web technology to feature modeling. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, pages 1252–1256. ACM, 2009. ISBN 978-1-60558-166-8. doi: 10.1145/1529282.1529563. (pages 5, 29, 32, 33).

[101] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308, pages 115–130. Springer–Verlag, 2004. doi: 10.1007/b102837. (pages 28, 29).

[102] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, June 2006. doi: 10.1007/s00766-006-0033-x. (page 28).

[103] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A bdd-based approach to verifying clone-enabled feature models' constraints and customization. In H. Mei, editor, *High*

Confidence Software Reuse in Large Systems, volume 5030 of *Lecture Notes in Computer Science*, pages 186–199. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-68062-8. (page 27).